**Project Report – Baris Alan**

**A. Key Findings**

1. Research and Development of LLMs: This study focuses on the research and development of large language models (LLMs), specifically examining how model size and the number of parameters impact performance. It aims to identify and recommend the most effective LLMs based on these factors.

**Objective**: To collect and analyze information on various open-source large language models (LLMs) by comparing their parameters, sizes, and general test results in a single table. This comparison aims to provide insights and recommendations for selecting the most suitable model based on these criteria.

https://github.com/Falgun1/NLP-Corpus/blob/main/Weekly%20Progress/Baris/Documents/model%20sizes.do

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Model** | **GLUE Score** | **SuperGLUE Score** | **MMLU Score** | **Code Generation Score** | **Overall Score** | **Parameter** | **Training Data Size (Tokens)** |
| LLaMA 3 - 70b | 86.5 | 84.5 | 83 | 81.5 | 83.9 | 70 billion | 15 Trillion |
| PaLM 2 | 86 | 84 | 82.5 | 80 | 83.1 | 340 billion | 3.6 trillion |
| LLaMA 2 - 70b | 85 | 83.5 | 82 | 79 | 82.4 | 70 billion | 2 trillion |
| Vicuna | 84 | 82.5 | 81 | 78 | 81.4 | 13 billion | 2048 |
| LLaMA 3 - 8b | 82 | 80.5 | 79 | 76 | 79.4 | 8 billion | 15 Trillion |
| LLaMA 2 - 13b | 82.5 | 81 | 79.5 | 76.5 | 79.9 | 13 billion | 2 trillion |

A screenshot of a phone

Description automatically generated

A screenshot of a screenshot of a graph

Description automatically generated

A screenshot of a graph

Description automatically generated

A screenshot of a graph

Description automatically generated

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Model** | **GLUE Score** | **SuperGLUE Score** | **MMLU Score** | **Code Generation Score** | **Overall Score** | **Parameter** | **Training Data Size (Tokens)** |
| LLaMA 2 - 7b | 80.5 | 79.5 | 78 | 75 | 78.3 | 7 billion |  |
| LLaMA 2 - 7b-instruct | 80 | 79 | 77.5 | 74.5 | 77.8 | 7 billion (fine-tuned) |  |
| Electra | 80 | 78.5 | 77 | Not ranked | 78.5 | 110 million |  |
| XLNet | 81 | 80 | 78.5 | Not ranked | 79.8 | 340 million |  |
| RoBERTa | 82.5 | 81.5 | 80.5 | Not ranked | 81.5 | 355 million |  |
| DeBERTa | 82 | 81 | 79.5 | Not ranked | 80.8 | 340 million |  |
| UniLM | 81 | 80 | 78.5 | Not ranked | 79.8 | 340 million |  |
| GPT-2 | 82 | 80.5 | 79 | Not ranked | 80.5 | 1.5 billion |  |
| CTRL | 79.5 | 78 | 76.5 | Not ranked | 78 | 110 million | 140GB Text Data |
| ERNIE | 80 | 78.5 | 77.5 | Not ranked | 78.7 | 340 million |  |
| BERT | 80.5 | 79.5 | 78 | Not ranked | 79.3 | 110 million |  |
| StableLM | 79 | 77.5 | 76 | 73 | 76.4 | 7 billion | 1.5 Trillion |
| Flan-T5 | 81.5 | 80 | 78.5 | 75.5 | 78.9 | 11 billion |  |
| BLOOM | 79 | 77.5 | 76 | 73 | 76.4 | 8 billion |  |
| T5 | 82 | 81 | 80 | Not ranked | 81 | 11 billion |  |
| ALBERT | 81.5 | 80.5 | 79.5 | Not ranked | 80.5 | 11 billion |  |

A diagram of a diagram

Description automatically generated

A screenshot of a phone

Description automatically generated

**Parameter Count:**

The number of parameters determines the size and complexity of the model. More parameters indicate a larger model.

Larger models can represent more information and learn more complex relationships.

However, more parameters also require more computational power and memory.

**Performance and Generalization:**

More parameters generally lead to better performance, especially with large datasets.

However, there is a risk of overfitting. Too many parameters can cause the model to memorize the training data and reduce generalization ability.

**Computational Power and Memory:**

More parameters require more computational power and memory. This is crucial during training and inference.

Larger models demand more CPU and memory resources.

**Censorship and Constraints:**

Fewer parameters may result in less censorship and fewer constraints. This allows the model to produce freer responses.

--------------------------------------------------------------------------------------

**Token Count:**

The number of tokens is an important factor that affects a language models performance and generalization ability.

**Few Tokens:**

Having fewer tokens results in faster inference times. The model can respond more quickly.

However, a smaller context is represented, containing less information. This may lead the model to provide narrower responses.

**Many Tokens:**

More tokens represent more information. This helps the model understand a broader context.

However, having too many tokens increases computational power and memory requirements. Larger models demand more resources.

**Risk of Overfitting:**

More tokens can increase the risk of overfitting to the training data. The model might memorize the training data and reduce generalization ability.

**Data Coverage:**

More tokens represent a wider range of data. This helps the model have knowledge across various topics.

In summary, finding the right balance of token count is essential for performance, speed, and generalization ability. The ideal token count depends on the specific use case and available resources

**LlamaIndex**

A diagram of a computer

Description automatically generated

I think it is a good way to start Minimum Viable Product when I search many methods for question generation. Firstly, I used LlamaIndex to convert the pdf file into the appropriate form and obtained the text document. Then I compared the results produced by different models, this will help to decide which model is preferred. After the question generation, answers will be sought with LlamaIndex.

A screenshot of a computer

Description automatically generated

A screen shot of a computer code

Description automatically generatedA computer screen shot of a black screen

Description automatically generatedA black rectangle with white text

Description automatically generatedA black screen with white text

Description automatically generated

A screenshot of a computer program

Description automatically generated

A screenshot of a computer program

Description automatically generated

A screen shot of a computer program

Description automatically generated

A screen shot of a computer screen

Description automatically generated

I applied for the same process for BART and shared the results below.

A close-up of a text

Description automatically generated

**Second Process**

**A diagram of a computer

Description automatically generated**

A screenshot of a computer screen

Description automatically generated

A screen shot of a computer

Description automatically generated

A screen shot of a computer code

Description automatically generated

A screen shot of a computer

Description automatically generated

The above-mentioned data collection is done by extracting from pdf to text, after the data collection process, data can also be collected from the visuals as appropriate.

Evaluating data Generation methodologies for fine-tuning LLM

Summary of Project: The primary goal of the project is to explore and evaluate different data generation methodologies for fine-tuning a large language model (LLM). The aim is to determine which methodology yields the best performance based on specific evaluation metrics.

Objectives

* Explore various data generation methodologies for the fine-tuning of LLMs.
* Evaluate the fine-tuned LLMs using specific performance metrics.
* Determine the best data generation methodologies for fine-tuning an LLM.

Scope

* Research available open-source datasets and LLMs.
* Develop code to run and fine-tune LLMs.
* Generate data using different methodologies for fine-tuning.
* Fine-tune the LLM using generated datasets.
* Evaluate the performance of the fine-tuned LLMs.

Expected Outcomes

* Working code for automatic data generation for fine-tuning an LLM.
* Working code for running and fine-tuning an LLM.
* Analysis and comparison of different data generation methodologies.
* Fine-tuned LLM models with performance evaluations.
* Documentation and reports summarizing the findings and methodologies.

Data

* Use public datasets related to the nuclear industry, such as books, research papers, articles, journals, etc.
* Ensure the datasets are free for commercial use.

Methodologies/ plan of attack

* 1. Perform research on Data, and LLMs (1 weeks)

Tasks:

* + - Identify available open-source datasets related to nuclear industry.
    - Investigate methods to collect the data efficiently.
    - Identify latest available open-source LLMs such as NuclearN.ai, LLAMA, Grok, etc.
    - Access the performances evaluation criteria of LLMs such as MMLU, TruthfulQA, etc while selecting an LLM.
    - Assess the data format and requirements for fine-tuning LLMs.
    - Access the computational resources required for inferencing, fine-tuning, etc.
    - Ensure data and models are free for commercial use.

Outcomes:

* + - Quick progress update and feedback session.
    - Selected dataset resources.
    - Selected a few suitable LLMs for the project.
    - Identified the specific fine-tuning requirements for the chosen model.
    - A written summary of findings for every step.
  1. Gather data (0.5 week)

Tasks:

* + - Collect the data using the identified methods.

Outcomes

* + - Collected data with a good directory structure.
  1. Develop code to run Open-Source non-fine tuned LLM (0.5 week)

Tasks:

* + - Obtain the model files from a trusted source or repository (e.g., Hugging Face, GitHub).
    - Set-up virtual environment.
    - Install necessary libraries and frameworks.
    - Develop inference logic to generate responses from the model.

Outcomes:

* + - An inference ready LLM integrated into your environment, capable of processing and responding to user inputs.
  1. Data Generation for fine-tuning LLM (3 weeks)

Tasks:

* + - Research and identify different methodologies for generating fine-tuning data from the selected dataset.
    - Ensure that the data generation methodologies align with the requirements of the chosen LLM.
    - Develop code to implement data generation methodologies.

Outcomes:

* + - Script(s) that can be used to generate data for fine tuning LLM utilizing the selected methodologies.
    - Multiple directories, one for every selected data generation methodology, containing data in proper format for fine-tuning LLM.
  1. Fine-tuning LLM (2 weeks)

Tasks:

* + - Research and identify different techniques to fine-tune an LLM.
    - Fine-tune the chosen LLM model on each of the different datasets.
    - Document the process, challenges, and solutions encountered during fine-tuning.

Outcomes:

* + - Fine-tuned models, one for every selected data methodology.
    - Detailed documentation of the fine-tuning process and challenges.
  1. Model Evaluation (1 week)

Tasks:

* + - Identify and select appropriate metrics for evaluating the performance of the fine-tuned LLM models.
    - Evaluate each fine-tuned model based on the selected metrics.
    - Perform a comparative analysis of the models fine-tuned with different data generation methodologies.
    - Analyze the evaluation results to determine which data generation methodology produced the best performance.
    - Offer a detailed explanation and justification for the chosen methodology based on the evaluation metrics.
    - Suggest potential improvements or future work.

Outcomes:

* + - Evaluation reports for each fine-tuned model.
    - Comparative analysis report highlighting the strengths and weaknesses of each data generation methodology.
    - Recommendations for the best data generation methodology based on the evaluation metrics.

General Guidance

Project Management

Effective project management requires using a task management dashboard, such as GitHub project or Azure DevOps, to track tasks and milestones.

Roles and responsibilities must be clearly defined and assigned for effective problem tackling. A person should be identified as a project manager lead, to recommend backlog items and assign to each person. As well as updating tasks onto a PM tool.

Regular update meetings should be held weekly to review progress, address issues, and adjust plans as necessary. The meeting will be led from the identified PM of the group, and the supervisor will provide further guidance and confirmation to the team.

Coding Practices

Code should be organized into clear, functional modules to enhance readability and maintainability. Following a consistent coding style and naming conventions is crucial for a uniform codebase.

Some general considerations:

1. Make modular code by having classes and functions: Utilize classes and functions to encapsulate functionality, promote reusability, and organize code logically.
2. Create a virtual environment for your project, keep track of the list of libraries and keep the versions consistent. Consider use tools like “pipenv” to manage the libraries and collaborate with team.
3. Include docstrings: Provide comprehensive docstrings for all functions, methods, and classes to describe their purpose, parameters, and return types.

* Docstrings should include a quick **description** of the function, **args**, and **output**.

1. Include type definitions: Define types for function parameters and return values to improve code clarity and help catch errors early.
2. Adopt script writing: Structure scripts to separate data processing, analysis, and utility functions into distinct modules, making the code easier to navigate and maintain.

Version Control

Using **GitHub** for version control is essential for managing code changes and collaboration.

Initially, a public repository needs to be created and can be used for access and collaboration, but it should be migrated to a private repository for production to ensure security and control.

Adopting a clear branching strategy; For team members contributing to the code, either fork the repository or make a separate branch to avoid conflicts. Meaningful commit messages that describe the changes made should be written to aid in understanding the project history.

Pull requests should be used to review and discuss code changes before merging them into the main branch.

Testing and Deployment

Robust testing and deployment practices are critical for ensuring the reliability and scalability of the application.

Unit tests should be written for individual functions and modules using frameworks such as pytest. Efficient testing/unit-tests is a integral part of the final product.

tests should ensure that different parts of the application work together as expected.

If time permits, setting up continuous integration (CI) pipelines to automatically run tests and checks on code commits helps maintain code quality. Continuous deployment (CD) pipelines should automate the deployment process to various environments, streamlining the release process.

Documentation

Code Documentation

Start with a comprehensive **README** file at the root of the project repository, providing an overview of the project, its purpose, main features, and instructions for setting up the development environment. Include a list of prerequisites, such as software dependencies and system requirements.

Methodology Documentation

Create a **word document** at the beginning, serving as a dynamic tracker, and documentation for the project. As the project progresses, continue to update the documentation with findings from experiments, analyses, and testing. Include detailed records of methodologies used, such as data collection processes, analysis techniques, and tools employed. Document any challenges encountered and how they were addressed, along with any changes in strategy or scope.

This document will ensure that lead, and future teams can quickly get up to speed and understand the project's evolution.

TYPES OF LLMs

**Autoregressive Language Models**:

* These models are trained to predict the next token (word or subword) in a sequence, given the previous tokens.
* Examples:
  + **GPT-2**: Developed by OpenAI, GPT-2 is a state-of-the-art natural language processing model. It's a large-scale unsupervised language model which means it can generate human-like text based on the input it receives. GPT-2 is trained on a massive amount of text data and is capable of tasks like text completion, summarization, translation, and more. It became well-known for its impressive ability to generate coherent and contextually relevant text.
    - Data Type: Text
    - Data Format: Tokenized text sequences, likely preprocessed from various formats (e.g., HTML, PDFs, plain text)
    - Evaluation: Perplexity
  + **GPT-Neo**: GPT-Neo, also known as GPT-NeoX-20B, is a large language model developed by EleutherAI, a collective of AI researchers and engineers. It is based on the GPT architecture and is trained on a diverse corpus of text data. GPT-Neo is designed to be an open-source alternative to GPT-3, aiming to provide a more accessible and transparent language model. With 20 billion parameters, it is one of the largest publicly available language models.
    - Data Type: Text
    - Data Format: Tokenized text sequences from the Pile dataset
  + **OPT**: OPT is a series of large language models developed by Meta AI (formerly Facebook AI Research). It is similar in architecture to GPT-3 but is trained on a different dataset and uses different techniques for pre-training. OPT models are available in various sizes, ranging from OPT-125M (125 million parameters) to OPT-66B (66 billion parameters). The OPT models are designed to be open and accessible, with the goal of enabling research and development in natural language processing.
    - Data Type: Text
    - Data Format: Tokenized text sequences from various online sources

**Masked Language Models**:

* These models are trained to predict masked (or missing) tokens in a sequence, given the surrounding context.
* Examples:
  + **BERT** : BERT is a transformer-based language model developed by Google AI in 2018. It is pre-trained on a large corpus of text data using a novel technique called "Masked Language Modeling" (MLM), where a certain percentage of the input tokens are randomly masked, and the model learns to predict the masked tokens based on the context. BERT is bidirectional, meaning it can process the input sequence from left to right and right to left simultaneously, allowing it to better capture context and meaning. BERT has been widely adopted and fine-tuned for various NLP tasks, such as text classification, question answering, and named entity recognition.
    - Data Type: Text
    - Data Format: Tokenized text sequences from various sources (e.g., Wikipedia, Books Corpus)
  + **RoBERTa**: RoBERTa is a variant of BERT developed by researchers at Facebook AI in 2019. It is trained on a larger dataset than the original BERT model and uses different training techniques, such as dynamic masking and longer training cycles. RoBERTa also removes the next-sentence prediction objective used in BERT's pre-training, focusing solely on the Masked Language Modeling task. These modifications aim to improve the model's performance and robustness across various NLP tasks.
    - Data Type: Text
    - Data Format: Tokenized text sequences from various online sources
  + **AlBERTv2**: AlBERTv2 is an improved version of the original ALBERT model, developed by researchers at Google in 2020. ALBERT is a lightweight variant of BERT, designed to reduce the model's memory footprint and increase training efficiency while maintaining competitive performance. AlBERTv2 introduces several improvements over the original ALBERT, including a better masking strategy, a more effective pre-training task, and a larger model size. These enhancements aim to improve the model's performance on various NLP tasks while maintaining its efficiency advantages.
    - Data Type: Text
    - Data Format: Tokenized text sequences from various online sources

**Sequence-to-Sequence Language Models**:

* These models are trained to generate an output sequence given an input sequence, useful for tasks like translation, summarization, and question-answering.
* Examples:
  + **BART** : BART is a sequence-to-sequence transformer model developed by researchers at Facebook AI in 2019. It is pre-trained on a large corpus of text data using a combination of two objectives: a bidirectional encoder objective (like BERT) and an auto-regressive decoder objective (like GPT). This bidirectional encoder-decoder architecture allows BART to handle a wide range of natural language generation tasks, such as machine translation, summarization, question answering, and text generation.
    - Data Type: Text
    - Data Format: Tokenized text sequences from various sources (e.g., news articles, books, websites)
  + **T5**: T5 is a unified transformer model developed by researchers at Google AI in 2019. It is designed to handle a wide range of natural language tasks by framing them as text-to-text problems. T5 is pre-trained on a massive corpus of text data using a simple yet effective objective: predicting the target text given the input text and a task description
    - Data Type: Text
    - Data Format: Tokenized text sequences from various online sources

**Multimodal Language Models**:

* These models can process and generate data in multiple modalities, such as text, images, and audio.
* Examples:
  + **DALL-E**: DALL-E is a deep learning model developed by OpenAI for generating images from text descriptions. It is a transformer-based neural network that is trained on a massive dataset of text-image pairs, allowing it to understand the relationship between natural language and visual representations.
    - Data Type: Text and Images
    - Data Format: Tokenized text sequences and image data (format not disclosed)
  + **Perceiver IO**: Perciver.IO is a deep learning model developed by researchers at DeepMind. It is a type of transformer model designed for multi-modal perception, which means it can process and learn from different types of data, such as images, text, and audio, simultaneously.
    - Data Type: Text, Images, Audio
    - Data Format: Tokenized text sequences, image data, and audio data (formats not disclosed)

**Instruction-Following Language Models**:

* These models are specifically designed to follow instructions and complete tasks based on natural language prompts.
* Examples:
  + **InstructGPT**: nstructGPT is trained using what's called "Cooperative AI" principles, which aim to make the model more aligned with human values and instructions. During training, the model is exposed to a large number of prompts that require it to follow specific instructions, guidelines, and constraints. This training procedure encourages the model to be more capable of understanding and adhering to complex instructions, while also promoting desirable traits like truthfulness, safety, and ethical reasoning.
    - Data Type: Text
    - Data Format: Tokenized text sequences from various online sources
  + **Claude**: Claude is an AI assistant created by Anthropic, and it is powered by the InstructGPT language model. Claude is designed to be a helpful and capable assistant that can engage in open-ended conversations, answer questions, and assist with a wide range of tasks, all while adhering to the user's instructions and following ethical principles.
    - Data Type: Text
    - Data Format: Tokenized text sequences from various online sources

**Retrieval-Augmented Language Models**:

* These models combine language modeling with retrieval from external knowledge sources, enabling them to leverage additional information beyond their training data.
* Examples:
  + **RAG**: RAG is a framework proposed by researchers at Facebook AI Research (FAIR) for improving the performance of language models on open-ended generation tasks, such as question answering and open-domain dialogue. The key idea behind RAG is to augment the language model with a separate retrieval component that can retrieve relevant information from a large corpus of text (e.g., Wikipedia).
    - Data Type: Text
    - Data Format: Tokenized text sequences from various sources (model and knowledge base)
  + **FiD**: FiD is an extension of the RAG framework, also proposed by researchers at FAIR. While RAG simply concatenates the retrieved passages with the input query, FiD goes a step further by fusing the retrieved information with the language model's generation process at a deeper level.
    - Data Type: Text
    - Data Format: Tokenized text sequences from various sources (model and knowledge base)

DATA TYPES AND DATA FORMATS

1. Data Types
2. **Text**: This is the primary data type that LLMs work with. They are trained on massive amounts of textual data from various sources like books, websites, articles, and databases.
3. **Tokenized Text**: Before processing text, LLMs often tokenize the text into smaller units called tokens, which can be individual words, subword units, or even individual characters. These tokens are then mapped to numerical values, which are the actual inputs to the LLM.
4. **Numerical Data**: Some LLMs can process numerical data, such as tabular data or time series data, by converting the numbers into textual representations or by using specialized architectures designed for handling numerical data.
5. **Images**: Some recent LLMs, such as Stable Diffusion and DALL-E, can process and generate images by treating them as sequences of pixels or tokens, similar to how they process text.
6. **Audio**: While not as common, some LLMs can process audio data by converting it into text (using automatic speech recognition) or by using specialized architectures designed for processing audio signals directly.
7. **Structured Data**: LLMs can process structured data formats like JSON, XML, or HTML by converting them into text or by using specialized architectures designed for handling such data.
8. Data Formats
   1. **Plain Text**: This is the most common data format for LLMs. They are trained on massive amounts of plain text data from various sources like books, websites, articles, and databases.
   2. **Tokenized Text**: Before processing text, LLMs often tokenize the text into smaller units called tokens. These tokens can be represented in various formats, such as integer sequences, byte-pair encoding (BPE), or other custom tokenization schemes.
   3. **JSON (JavaScript Object Notation)**: JSON is a lightweight data-interchange format that is widely used for representing structured data. LLMs can process and generate JSON data, which can be useful for tasks like knowledge base construction, data augmentation, or generating structured outputs.
   4. **XML (Extensible Markup Language)**: XML is another format for representing structured data. LLMs can process and generate XML data, which can be useful for tasks like document generation, data extraction, or working with markup languages.
   5. **HTML (Hypertext Markup Language)**: HTML is a markup language used for creating web pages. LLMs can process and generate HTML data, which can be useful for tasks like web content generation, data extraction from web pages, or generating markup for web applications.
   6. **CSV (Comma-Separated Values)**: CSV is a simple file format used for storing tabular data, such as spreadsheets or databases. LLMs can process and generate CSV data, which can be useful for tasks like data analysis, data augmentation, or generating tabular outputs.
   7. **Image Formats (PNG, JPEG, etc.)**: Some LLMs, particularly those used for multimodal tasks, can process and generate image data in various formats like PNG, JPEG, or BMP. These LLMs treat images as sequences of pixels or tokens, similar to how they process text.
   8. **Audio Formats (WAV, MP3, etc.)**: While less common, some LLMs can process and generate audio data in formats like WAV or MP3. These LLMs typically convert the audio data into a textual or numerical representation before processing.
   9. **Serialization Formats (pickle, protobuf, etc.)**: LLMs can work with various serialization formats, which are used to store and transmit structured data. These formats can be useful for tasks like model persistence, data exchange, or working with structured data types.

**Question-Answer (Q&A) Format:**

In your training dataset, you should have pairs consisting of questions related to nuclear energy and their corresponding correct answers. Each pair should include one question and its corresponding correct answer. In your test dataset, you should have similar question-answer pairs to assess the accuracy of the model. However, the correct answers should be kept hidden in the test dataset to evaluate the real-world performance of the model.

**Key metrics for Q&A evaluation:**

* **Response relevance:** Assess how well the system's answers align with the query's context and intent.
* **Sentiment analysis:** Evaluate the emotional tone of both queries and responses, ensuring appropriateness for customer interactions.
* **Content compliance:** Monitor for "jailbreak" instances where responses deviate from expected norms or rules, ensuring content remains on-topic and within ethical guidelines.
* **Toxicity detection:** Implement checks for harmful or offensive language to maintain a safe interaction environment.
* **Accuracy:** Measure the correctness of the responses provided by the system compared to the ground truth.
* **Precision and Recall:** Evaluate the trade-off between providing relevant responses (precision) and capturing all relevant responses (recall).
* **Fluency:** Assess the naturalness and coherence of the system's responses, ensuring they are grammatically correct and flow logically.
* **Engagement:** Measure user engagement metrics such as click-through rates, session duration, and repeat interactions to gauge the effectiveness of the Q&A system.
* **Robustness:** Test the system's performance under various conditions, including noisy input, ambiguous queries, and language variations, to ensure reliability in real-world usage.

**Dialogue Format:**

In your training dataset, you should have texts containing dialogues between users and the model. Each dialogue can involve users asking questions about nuclear energy, and the model responding to these questions. In your test dataset, you should include similar dialogues to evaluate how the model performs in real-world scenarios.

**Key metrics for Dialogue evaluation:**

* **Dialogue flow:** Evaluate the coherence and natural progression of the conversation between the user and the system.
* **User satisfaction:** Measure user feedback and sentiment during and after the dialogue to gauge satisfaction with the system's responses.
* **Task completion:** Measure the system's ability to fulfill user requests and assist in accomplishing tasks effectively within the dialogue.
* **Politeness and empathy:** Evaluate the system's use of polite language and empathetic responses to maintain a positive interaction experience.
* **Error handling:** Assess the system's ability to gracefully handle errors, misunderstandings, and unexpected inputs during the dialogue.
* **Conversational depth:** Measure the depth of the conversation in terms of the variety of topics covered and the complexity of interactions.
* **User retention:** Track user retention rates and repeat visits to evaluate the system's ability to maintain user interest and encourage return interactions.

**Example sites;** <https://huggingface.co/datasets/nvidia/HelpSteer>

https://arxiv.org/pdf/2306.13304

A screenshot of a black and white screen

Description automatically generated

A screenshot of a computer

Description automatically generated

**Text Data:**

* **Format:** Plain text files (.txt), JSON files, or CSV files.
* **Structure:** Each line or row represents a separate text document or sentence.
* **Encoding:** UTF-8 encoding is commonly used to support a wide range of characters and languages.
* **Preprocessing:** Text may be preprocessed to remove special characters, punctuation, and stopwords, and perform tokenization and normalization.

**Structured Data:**

* **Format:** CSV files, JSON files, or database tables (e.g., SQL databases).
* **Structure:** Tabular format with rows and columns, where each row represents a data instance and each column represents a feature or attribute.
* **Fields:** Column headers represent feature names, while rows contain corresponding values.
* **Encoding:** UTF-8 encoding for text fields, numeric encoding for numerical fields, and appropriate encoding for categorical fields (e.g., one-hot encoding or label encoding).

EVALUATION METRICS

1. **Perplexity**: Perplexity is a measure of how well a probability model (in this case, an LLM) predicts a sample of text. Lower perplexity scores indicate better performance, as the model is more confident in its predictions.
2. **Cross-Entropy Loss**: Cross-entropy loss is a metric commonly used for evaluating language models on sequence prediction tasks. It measures the performance of a model in predicting the next token in a sequence, given the previous tokens.
3. **BLEU (Bilingual Evaluation Understudy)**: BLEU is a widely used metric for evaluating machine translation systems, but it can also be applied to evaluate the quality of text generated by LLMs. It measures the overlap between the generated text and one or more reference texts.
4. **ROUGE (Recall-Oriented Understudy for Gisting Evaluation)**: ROUGE is a set of metrics used for evaluating automatic summarization systems, but it can also be applied to evaluate the quality of text generated by LLMs in tasks like abstractive summarization or question answering.
5. **METEOR (Metric for Evaluation of Translation with Explicit ORdering)**: METEOR is another metric used for evaluating machine translation systems, but it can also be applied to evaluate the quality of text generated by LLMs. It considers not only the overlap between the generated text and reference text but also the matching of longer phrases and the reordering of words.
6. **BERTScore**: BERTScore is a metric that leverages pre-trained language models (like BERT) to evaluate the quality of text generated by LLMs. It measures the semantic similarity between the generated text and reference text.
7. **Human Evaluation**: In some cases, human evaluation is used to assess the quality of text generated by LLMs. This can involve tasks like rating the coherence, fluency, or relevance of the generated text.
8. **Task-Specific Metrics**: Depending on the specific task or application, task-specific metrics may be used to evaluate LLMs. For example, accuracy, precision, recall, and F1-score for classification tasks, or mean reciprocal rank (MRR) for information retrieval tasks.
9. **Automated Metrics**: Various automated metrics, such as GLEU (Google-BLEU), NIST (National Institute of Standards and Technology), or TER (Translation Edit Rate), can also be used to evaluate LLMs on specific tasks or domains.
10. **Accuracy:** Accuracy measures how often a model produces correct results for a given task. For example, in a classification task, accuracy shows how frequently the model predicts the correct labels.

In real-world applications, accuracy is crucial as it often influences decision-making processes. Therefore, models exhibiting "Medium" accuracy levels might suffice in many scenarios, but certain use cases may require higher accuracy.

1. **F1 Score:**F1 Score is the harmonic mean of precision and recall. It is used to achieve balanced results in classification tasks.

The F1 score measures the balance between precision and recall, particularly important in classification tasks where balanced results are desired. While "Medium" F1 scores might be acceptable in many applications, higher F1 scores might be required in certain scenarios.

1. **MMLU (Mean Max Log Likelihood):** Mean Max Log Likelihood measures how well a language model predicts a particular text. A higher MMLU value indicates that the model predicts the text better.
2. **TruthfulQA Score:** TruthfulQA is an accuracy metric that measures the accuracy and coverage, determining whether correct answers are found.

In real-world applications, accuracy is crucial for a model's reliability. Therefore, a "Medium" TruthfulQA score indicates the model's ability to provide correct answers to specific questions.

1. **SQuAD Score:** Stanford Question Answering Dataset (SQuAD) is used to evaluate model performance in text-based question answering tasks. A high SQuAD score indicates better question-answer matching.

**Accuracy:**

Accuracy measures how often a model produces correct results for a given task. For example, in a classification task, accuracy shows how frequently the model predicts the correct labels.

In real-world applications, accuracy is crucial as it often influences decision-making processes. Therefore, models exhibiting "Medium" accuracy levels might suffice in many scenarios, but certain use cases may require higher accuracy.

**F1 Score:**

F1 Score is the harmonic mean of precision and recall. It is used to achieve balanced results in classification tasks.

The F1 score measures the balance between precision and recall, particularly important in classification tasks where balanced results are desired. While "Medium" F1 scores might be acceptable in many applications, higher F1 scores might be required in certain scenarios.

**BLEU Score:**

BLEU (Bilingual Evaluation Understudy) is a metric used to evaluate translation quality. It measures how closely the generated translation matches the translations produced by human translators.

The BLEU score, evaluating translation quality, is important in text translation applications. Low BLEU scores may indicate that the translation does not resemble those produced by human translators. Hence, a "Low" BLEU score could be concerning in terms of translation quality.

**Rouge Score:**

Rouge (Recall-Oriented Understudy for Gisting Evaluation) is a metric used in text summarization tasks. It measures how similar the generated summary is to the reference summary (e.g., summaries created by humans).

The Rouge score, used in text summarization tasks, measures how closely the generated summary matches the reference summary. Low Rouge scores may indicate poor summarization abilities of the model.

**Perplexity:**

Perplexity measures a language model's ability to predict a given text. A lower perplexity value indicates that the model predicts the text better.

Perplexity, measuring how well a language model predicts text, reflects the model's quality. A lower perplexity indicates better language understanding abilities.

**MMLU (Mean Max Log Likelihood):**

Mean Max Log Likelihood measures how well a language model predicts a particular text. A higher MMLU value indicates that the model predicts the text better.

**TruthfulQA Score:**

TruthfulQA is an accuracy metric that measures the accuracy and coverage, determining whether correct answers are found.

In real-world applications, accuracy is crucial for a model's reliability. Therefore, a "Medium" TruthfulQA score indicates the model's ability to provide correct answers to specific questions.

**SQuAD Score:**

Stanford Question Answering Dataset (SQuAD) is used to evaluate model performance in text-based question answering tasks. A high SQuAD score indicates better question-answer matching.

**GLUE Score:**

General Language Understanding Evaluation (GLUE) is a metric used to evaluate model performance in language understanding tasks. A low GLUE score may indicate that the model's overall language comprehension ability is poor and that he/she fails in language comprehension tasks.

**Parameter Count:**

The number of parameters determines the size and complexity of the model. More parameters indicate a larger model.

Larger models can represent more information and learn more complex relationships.

However, more parameters also require more computational power and memory.

**Performance and Generalization:**

More parameters generally lead to better performance, especially with large datasets.

However, there is a risk of overfitting. Too many parameters can cause the model to memorize the training data and reduce generalization ability.

**Computational Power and Memory:**

More parameters require more computational power and memory. This is crucial during training and inference.

Larger models demand more CPU and memory resources.

**Censorship and Constraints:**

Fewer parameters may result in less censorship and fewer constraints. This allows the model to produce freer responses.

--------------------------------------------------------------------------------------

**Token Count:**

The number of tokens is an important factor that affects a language models performance and generalization ability.

**Few Tokens:**

Having fewer tokens results in faster inference times. The model can respond more quickly.

However, a smaller context is represented, containing less information. This may lead the model to provide narrower responses.

**Many Tokens:**

More tokens represent more information. This helps the model understand a broader context.

However, having too many tokens increases computational power and memory requirements. Larger models demand more resources.

**Risk of Overfitting:**

More tokens can increase the risk of overfitting to the training data. The model might memorize the training data and reduce generalization ability.

**Data Coverage:**

More tokens represent a wider range of data. This helps the model have knowledge across various topics.

In summary, finding the right balance of token count is essential for performance, speed, and generalization ability. The ideal token count depends on the specific use case and available resources

· **LLaMA 3 - 70B**:

* LLaMA 3 70B is a large language model containing approximately 15 trillion tokens.
* It performs highly and can generate intelligent responses.
* It has a lower censorship level and fewer restrictions.

· **PaLM 2**:

* PaLM 2 is created by fine-tuning the base LLaMA 2 model.
* It is smaller in size and capabilities compared to LLaMA 3 70B.

· **LLaMA 2 - 70B**:

* LLaMA 2 70B is a large language model with 70 billion tokens.
* It performs well and can understand a wide range of data similar to LLaMA 3 70B.

· **Vicuna**:

* Vicuna-13B is created by fine-tuning the base LLaMA 2 model.
* It performs very successfully and can surpass GPT-4 based models.

· **LLaMA 3 - 8B**:

* LLaMA 3 8B is smaller in size with approximately 8 billion tokens.
* It still performs quite well and has less censorship.

· **LLaMA 2 - 13B**:

* LLaMA 2 13B is a medium-sized version within the LLaMA 2 series.
* It can understand more nuances and is suitable for creating creative writings or poems.

**Llama 3 8B and Llama 3 70B** models represent significant advancements in the field of large language models, pushing the boundaries in terms of performance, scalability, and capabilities.

**Parameter Count**: The Llama 3 70B model has more parameters, making it a larger and more complex model. The Llama 3 8B model has fewer parameters and is smaller in size.

**Performance:** Generally, a model with more parameters tends to perform better. Therefore, the Llama 3 70B model can generate smarter and more consistent responses. However, the Llama 3 8B model, despite being an 8B model, performs quite well. It excels particularly in reducing repetitions and remembering past conversations compared to Llama 2 70B models.

**LLaMA 3** 70B can represent more information due to having more parameters. This means a larger model can learn more data.

However, the smaller size of Vicuna-13B might help it utilize training data more effectively. Fewer parameters could lead to faster training and better generalization.

Learning Capabilities: Large models can better grasp complex relationships, though this requires more data. Despite having fewer parameters, Vicuna-13B can still learn significant information efficiently from its training data.

Larger models can create more complex and extensive internal representations, albeit requiring more computational power. Due to its smaller size, Vicuna-13B may generate more limited internal representations, yet it can still achieve effective results