

Intelligent recommendation system using large language models for process safety improvement

BACHELOR TERM PROJECT REPORT

By

Abhishek Choudhary
(20ME31057)

Under the supervision of

Prof. J. MAITI
Department of Industrial and Systems Engineering, IIT Kharagpur



Department of Industrial and Systems Engineering
Indian Institute of Technology Kharagpur
West Bengal, India
30th April, 2024

DECLARATION

I declare that,

The work contained in this report has been done by me under the guidance of my supervisor. The work has not been submitted to any other Institute for any degree or diploma.

I have conformed to the norms and guidelines given in the Ethical Code of Conduct of the Institute.

Whenever I have used materials (data, theoretical analysis, figures, and text) from other sources, I have given due credit to them by giving their details in the references.

Date: 30-04-24
Place: Kharagpur

Name: Abhishek Choudhary
Roll number: 20ME3105

DEPARTMENT OF INDUSTRIAL AND SYSTEMS ENGINEERING
INDIAN INSTITUTE OF TECHNOLOGY, KHARAGPUR
KHARAGPUR-721302, INDIA



CERTIFICATE

This is to certify that the project report entitled “**Intelligent recommendation system using large language models for process safety improvement**” submitted by **Abhishek Choudhary (20ME31057)** to Indian Institute of Technology, Kharagpur towards partial fulfillment of requirements for the award of degree of Dual Bachelor of Technology in Mechanical Engineering is a record of bona fide work carried out by him under my supervision and guidance during Spring Semester 2023-24.

Date: 30/04/2024

Place: Kharagpur

Prof J Maiti

Department of Industrial and Systems Engineering

Indian Institute of Technology, Kharagpur

Kharagpur, India

CONTENTS

1. Introduction
2. Literature Review
3. Objectives
4. Methodology
5. Results
6. Discussion
7. Conclusion
8. References

1. Introduction

Industrial safety stands as an indispensable pillar of contemporary industrial practices, entrenched in the ethos of responsible business conduct and societal welfare. Within the intricate web of industrial operations, where machinery hums, chemicals flow, and human expertise navigates complex processes, the specter of potential hazards looms large. From manufacturing floors to petrochemical refineries, adherence to stringent safety protocols isn't merely a matter of compliance; it's a moral imperative, safeguarding lives, protecting the environment, and preserving the economic viability of enterprises.

Importance of Industrial Safety

The significance of industrial safety extends far beyond the confines of regulatory compliance; it's a linchpin of sustainable business operations. Robust safety measures not only shield workers from harm but also mitigate risks to infrastructure, equipment, and the environment. By fostering a culture of safety, organizations cultivate an environment where employees feel valued and empowered, leading to increased morale, productivity, and retention rates. Moreover, the tangible benefits of prioritizing safety are evident in reduced downtime, lower insurance premiums, and enhanced brand reputation, positioning safety as a strategic imperative rather than a mere operational necessity.

Introduction to Seq-to-Seq Modeling

Seq-to-seq models, a fundamental concept in natural language processing (NLP), are designed to process input sequences and generate corresponding output sequences. Traditionally, recurrent neural networks (RNNs) were used for seq-to-seq tasks, but they often struggled with capturing long-range dependencies and suffered from the vanishing gradient problem. Transformers, with their innovative architecture based on self-attention mechanisms, have revolutionized seq-to-seq modeling by addressing these limitations.

Transformers and Self-Attention Mechanisms

At the core of transformer models lie self-attention mechanisms, which enable the model to weigh the importance of different input tokens based on their relevance to each other. Unlike RNNs, which process sequences sequentially, transformers can capture long-range dependencies in the data more effectively by attending to relevant information dynamically. This dynamic attention mechanism allows transformers to focus on relevant parts of the input sequence, facilitating more nuanced understanding and generation of sequential information.

Multi-Head Attention and Positional Encoding

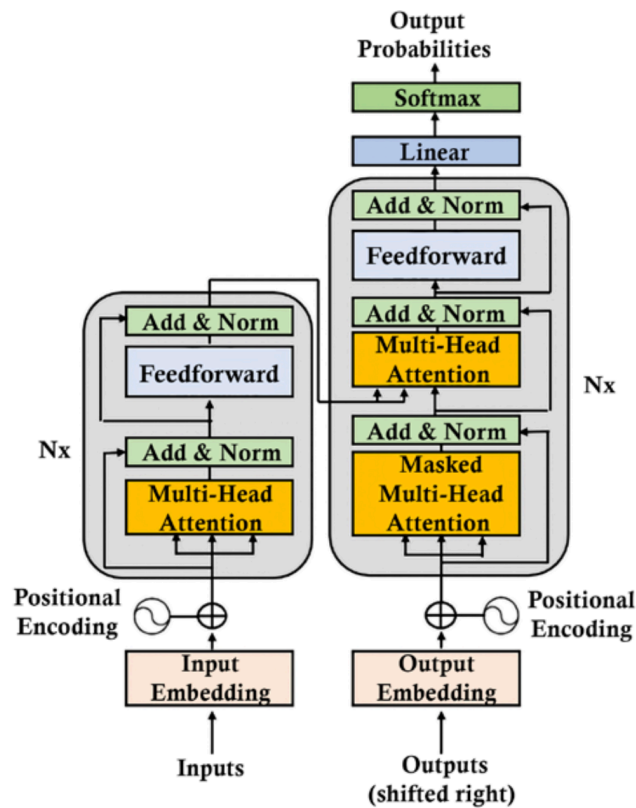
Transformers leverage multi-head attention mechanisms, where attention is computed multiple times in parallel, each focusing on different aspects of the input sequence. This multi-head attention enhances the model's ability to capture diverse patterns and relationships in the data, thereby improving its performance in seq-to-seq tasks. Additionally, transformers incorporate positional encoding to provide information about the position of tokens in the sequence, preserving sequential order and capturing temporal dependencies.

Applications of Seq-to-Seq with Transformers

Seq-to-seq tasks encompass a wide range of applications in NLP, including machine translation, text summarization, and recommendation generation. Transformers excel in these tasks due to their inherent ability to capture long-range dependencies and contextual information. In machine translation, transformers can encode the source language sequence and generate the target language sequence by attending to relevant parts of the input and leveraging contextual information to produce accurate translations. Similarly, in text summarization tasks, transformers can distill large volumes of text into concise summaries by identifying key information and generating coherent summaries based on the input sequence.

Advantages of Seq-to-Seq with Transformers

Seq-to-seq models with transformers offer several advantages over traditional approaches, including improved performance in capturing complex patterns and relationships in sequential data, better handling of long-range dependencies, and more efficient training and learning. Their ability to process input sequences holistically and generate contextually relevant outputs makes them indispensable tools for a wide range of NLP tasks, paving the way for new breakthroughs in language understanding, generation, and translation.



Types of Large Language Model (LLM) Architectures

Large Language Models (LLMs) have evolved to encompass various architectural paradigms, each designed to address specific challenges and tasks within natural language processing (NLP). These architectures have significantly advanced the state-of-the-art in NLP, enabling breakthroughs in tasks such as language understanding, generation, and translation. Here, we explore three prominent types of LLM architectures and their applications:

1. Decoder Models (e.g., GPT - Generative Pre-trained Transformer):

Decoder models are characterized by their unidirectional processing flow, where the model generates output sequences based solely on the input sequence. GPT, a pioneering example of a decoder model, employs a stack of transformer layers to generate coherent and contextually relevant text. These models excel in tasks such as text generation, summarization, and dialogue generation, where generating fluent and coherent sequences is paramount. Their ability to capture long-range dependencies and contextual information makes them particularly effective in tasks requiring sequential reasoning and generation.

Applications:

- **Text generation:** GPT can generate natural-sounding text in a variety of styles and genres, making it useful for applications such as content creation, chatbots, and creative writing assistance.
- **Summarization:** GPT can distill large volumes of text into concise summaries while preserving key information and semantic coherence.
- **Dialogue systems:** GPT can engage in natural and contextually relevant conversations, making it suitable for building chatbots and virtual assistants.

2. Encoder Models (e.g., BERT - Bidirectional Encoder Representations from Transformers):

Encoder models focus on encoding input sequences bidirectionally, capturing contextual information from both the left and right contexts. BERT, a prominent example of an encoder model, employs a transformer architecture to generate high-quality contextual embeddings for input text. These models excel in tasks such as sentence classification, question answering, and named entity recognition, where understanding the context and semantics of input sequences is critical.

Applications:

- **Sentence classification:** BERT can classify sentences into predefined categories, making it useful for sentiment analysis, topic classification, and spam detection.
- **Question answering:** BERT can generate accurate and contextually relevant answers to user queries by understanding the context of the question and the relevant information in the input text.
- **Named entity recognition:** BERT can identify and classify named entities such as persons, organizations, and locations in text, aiding in information extraction and semantic analysis.

3. Encoder-Decoder Models (e.g., BART - Bidirectional and Auto-Regressive Transformers):

Encoder-decoder models combine the strengths of both encoder and decoder architectures, allowing for bidirectional processing of input sequences and generation of output sequences. BART, an example of an encoder-decoder model, leverages a transformer-based architecture to perform tasks such as text generation, translation, and summarization. These models are versatile and can handle a wide range of seq-to-seq tasks requiring both understanding and generation of text.

Applications:

- **Text generation:** BART can generate coherent and contextually relevant text for various applications such as machine translation, text summarization, and paraphrasing.
- **Machine translation:** BART can translate text between different languages by encoding the source language sequence and decoding it into the target language sequence.
- **Summarization:** BART can generate concise and informative summaries of input text by encoding the input sequence and generating a summary sequence based on the encoded representations.

Types of Large Language Models (LLM)

Large language models (LLM) encompass a diverse array of architectures, each imbued with unique capabilities and characteristics. From autoregressive models like GPT to bidirectional models like BERT, LLMs differ in their pre-training objectives, architectural nuances, and fine-tuning strategies. Autoregressive models like GPT generate text sequentially, conditioning each word on the preceding context, facilitating coherent and contextually relevant text generation. On the other hand, bidirectional models like BERT leverage bidirectional context understanding, capturing contextual information from both directions to enhance understanding and performance in downstream tasks.

Fine-Tuning Models

Fine tuning Large-Language-Models

Supervised fine tuning (in 5 steps)

1. Choose a fine-tuning task
2. Prepare training dataset
3. choose a base model
4. Fine-tune model via supervised learning
5. Evaluate model performance

Fine-tuning is the process of taking a pre-trained model and adjusting at least one internal parameter (i.e., weights) during training. In the context of Large Language Models (LLMs), fine-tuning transforms a general-purpose base model (e.g., GPT-3) into a specialized model tailored for a specific use case (e.g., ChatGPT).

The primary advantage of fine-tuning is that it allows models to achieve improved performance with fewer manually labeled examples compared to models trained solely via supervised learning.

While base self-supervised models can perform well across various tasks with the aid of prompt engineering, they may still generate completions that are not entirely accurate or helpful. For instance, comparing completions between the base GPT-3 model (davinci) and a fine-tuned model (text-davinci-003) reveals that the fine-tuned model provides more helpful responses. The fine-tuning process used for text-davinci-003, known as alignment tuning, aims to enhance the helpfulness, honesty, and harmlessness of the LLM's responses.

The benefits of fine-tuning include:

- 1. Improved Performance:** A smaller, fine-tuned model can often outperform larger, more expensive models on specific tasks for which it was trained.
- 2. Efficiency:** Despite being smaller in size, fine-tuned models can achieve better task performance, as demonstrated by OpenAI's InstructGPT models, which outperformed the larger GPT-3 base model.
- 3. Domain-specific Knowledge:** Fine-tuning allows models to learn domain-specific information during the training process, mitigating the issue of finite context windows and improving performance on tasks requiring specialized knowledge.
- 4. Lower Inference Costs:** Fine-tuned models can reduce the need for additional context in prompts, resulting in lower inference costs.

3 Ways to Fine-tune Language Models

Fine-tuning a language model is a crucial step in leveraging its power for specific tasks. There are three primary approaches to fine-tuning: self-supervised learning, supervised learning, and reinforcement learning. These approaches can be used individually or in combination to enhance the performance of the model.

Self-supervised Learning

Self-supervised learning involves training a model based on the inherent structure of the training data. In the case of Language Model Models (LLMs), this often entails predicting the next word given a sequence of words or tokens. Self-supervised learning can be utilized to develop models that mimic a person's writing style using example texts.

Supervised Learning

Supervised learning is a widely used method for fine-tuning language models. It involves training the model on input-output pairs specific to a particular task. For instance, instruction tuning aims to enhance model performance in tasks like question answering or responding to user prompts. Creating question-answer pairs and integrating them into prompt templates is essential for supervised learning.

Reinforcement Learning

Reinforcement learning (RL) is another approach to fine-tuning language models. RL employs a reward model to guide the training process by scoring model completions based on human preferences. OpenAI's InstructGPT models exemplify how RL can be utilized for model fine-tuning through multiple steps involving supervised learning and RL algorithms like Proximal Policy Optimization (PPO).

Supervised Fine-tuning Steps (High-level)

For this article's purpose, we will delve into the supervised fine-tuning approach. Below is a high-level outline of the steps involved:

- 1. Choose fine-tuning task:** Select the specific task for fine-tuning, such as summarization, question answering, or text classification.
- 2. Prepare training dataset:** Create input-output pairs and preprocess the data by tokenizing, truncating, and padding text.
- 3. Choose a base model:** Experiment with different pre-trained models and select the one that best suits the task.
- 4. Fine-tune model via supervised learning:** Train the model using the prepared dataset and evaluate its performance.

While each step in the fine-tuning process is critical, we will primarily focus on the training procedure in this article. Understanding how to effectively train a fine-tuned model is essential for achieving optimal performance in various natural language processing tasks.

3 Options for Parameter Training

When fine-tuning a model with a substantial number of parameters, ranging from ~100M to 100B, it's essential to consider computational costs and the strategy for parameter training. This article explores three generic options for parameter training and discusses their implications.

Option 1: Retrain All Parameters

The simplest approach is to retrain all internal model parameters, known as full parameter tuning. While straightforward, this method is computationally expensive and may lead to catastrophic forgetting, where the model forgets useful information learned during initial training.

Option 2: Transfer Learning

Transfer learning (TL) involves preserving the useful representations/features learned by the model from past training when applying it to a new task. TL typically entails replacing the final layers of the neural network with new ones while leaving the majority of parameters unchanged. While TL reduces computational costs, it may not address the issue of catastrophic forgetting.

Option 3: Parameter Efficient Fine-tuning (PEFT)

Parameter Efficient Fine-tuning (PEFT) is a family of techniques aimed at fine-tuning a base model with a small number of trainable parameters. One popular method within PEFT is Low-Rank Adaptation (LoRA), which modifies a subset of layers' weights using a low-rank matrix decomposition technique. LoRA significantly reduces the number of trainable parameters while maintaining performance comparable to full parameter tuning.

Other popular and more efficient method is quantized-lora (or qlora). QLoRA is a fine-tuning technique that combines a high-precision computing technique with a low-precision storage method. It is one of the most memory and computationally efficient fine-tuning methods ever created. QLoRA works by transmitting gradient signals through a fixed, 4-bit quantized pre-trained language model into Low Rank Adapters (LoRA). The weights of a pre-trained language model are quantized to 4-bits using the NormalFloat encoding. This produces a 4-bit base model that retains the full model architecture but with weights stored in a compact quantized format.

Example Code 1: Fine-tuning an LLM using LoRA

We will demonstrate fine-tuning a language model using LoRA for sentiment classification. We'll use the Hugging Face ecosystem to fine-tune the `distilbert-base-uncased` model, a ~70M parameter model based on BERT, for classifying text as 'positive' or 'negative'. By employing transfer learning and LoRA, we can efficiently fine-tune the model to run on personal computers within a reasonable timeframe.

by using this PEFT method, we can reduce training parameters

```
>> trainable params: 628,994 || all params: 67,584,004 || trainable%: 0.9306847223789819
```

Example code 2: Fine-tuning an LLM using Qlora

we will demonstrate fine-tuning a model using qlora for summarizing the conversation. we'll be using hugging face ecosystem to fine-tune the NousResearch/Llama-2-7b-hf model, a ~7B parameter model based on llama 2. for summarizing the conversation between people. By employing transfer learning and QLoRA, we can efficiently fine-tune the model to run on personal computers within a reasonable timeframe.

by using this PEFT method, we can reduce training parameters

>> trainable params: 16777216 || all params: 3517190144 || trainable%: 0.477006226934315

--->>>> For detailed code examples and resources, please refer to the GitHub repository and Hugging Face for the final model and dataset.

<https://github.com/devProAbhi/Fine-tuning-LLM>

```
bnb_config = BitsAndBytesConfig(
    load_in_4bit=True,
    bnb_4bit_use_double_quant=True,
    bnb_4bit_quant_type="nf4",
    bnb_4bit_compute_dtype=torch.bfloat16
)
```



```

from peft import LoraConfig, get_peft_model

lora_config = LoraConfig(

    # r stands number of rank decomposition matrix (higher rank leads better results but high computations)
    r=16,

    lora_alpha=64, # scaling factor, determines the extend to which the model is adapted towards new training data
    #(lower value gives more weightage to the original data or biased towards base model)

    # target_modules=["query_key_value"],

    #specific to Llama models. here we can determines which specific weights and matrices are to be trained
    #(most basic are to train query and value vectors)

    target_modules=["q_proj", "k_proj", "v_proj", "o_proj"], #here o_proj is for output

    lora_dropout=0.1,
    bias="none",
    task_type="CAUSAL_LM"
)

model = get_peft_model(model, lora_config)
print_trainable_parameters(model)

```

trainable params: 16777216 || all params: 3517190144 || trainable%: 0.477006226934315

Evaluation

Evaluation constitutes a critical aspect of assessing the efficacy and performance of seq-to-seq models in real-world scenarios. Metrics such as BLEU (Bilingual Evaluation Understudy), ROUGE (Recall-Oriented Understudy for Gisting Evaluation), and perplexity provide quantitative measures of the quality and coherence of generated sequences, capturing aspects such as fluency, coherence, and relevance. Human evaluation, conducted through expert judgment or crowdsourcing, offers valuable qualitative insights into the model's effectiveness and potential areas for improvement, ensuring alignment with user expectations and domain-specific requirements. Additionally, ablation studies, where specific components or features of the model are systematically removed, can provide insights into the model's robustness and the relative importance of different components in achieving optimal performance.

```
[39] from transformers import TrainingArguments

training_arguments = TrainingArguments(
    per_device_train_batch_size=16,
    gradient_accumulation_steps=4,
    optim="paged_adamw_32bit", #a special type of adam optimizer used for training llm models
    logging_steps=1,
    learning_rate=1e-4,
    fp16=True,
    max_grad_norm=0.3,
    num_train_epochs=10,
    evaluation_strategy="steps",
    eval_steps=0.2,
    warmup_ratio=0.05,
    save_strategy="epoch",
    group_by_length=True,
    output_dir=OUTPUT_DIR,
    report_to="tensorboard",
    save_safetensors=True,
    lr_scheduler_type="cosine",
    seed=42,
)
model.config.use_cache = False # silence the warnings. Please re-enable for inference!
```

```
from trl import SFTTrainer
trainer = SFTTrainer(
    model=model,
    train_dataset=train_data,
    eval_dataset=validation_data,
    peft_config=lora_config,
    dataset_text_field="text",
    max_seq_length=1024,
    tokenizer=tokenizer,
    args=training_arguments,
)

trainer.train()
```

2. Literature Review

In 2011, Rivas et al. used machine learning techniques, including Bayesian Networks (BN) and Support Vector Machines (SVM), to model accidents in construction and mining

companies. BN-based prediction models have been applied in various sectors, such as mining and construction (Matías et al., 2008). For example, Sanmiquel et al. analyzed 69,869 workplace accidents in the Spanish mining sector from 2003 to 2012 using BN (San Miguel et al., 2015). In addition to modeling workplace accidents, extensive literature exists on various accident types, with road accidents receiving particular attention (Alikhani et al., 2013). Examining the accident analysis domain, popular and effective techniques include SVM and KNN, known for their robust theoretical foundations that facilitate successful data learning.

The majority of accident domain literature primarily relies on numerical or categorical data for analyzing accident scenarios. Yet, the exploration of free-text data, such as descriptive information, is often overlooked due to the challenging nature of extracting patterns from unstructured passages. Narrative text stands out as a crucial resource for predicting accident risk levels, offering valuable supplementary insights alongside other data types in the analysis.

In exploring the significance of narratives in predicting workplace accidents, Jones and Lyons demonstrated an uptick of 19% in home injuries, a substantial 137% increase in rugby injuries, and a 26% rise in assaults (Jones and Lyons, 2003). Additionally, Brown made a significant contribution by utilizing text mining, along with techniques such as Random Forest (RF), to analyze rail accident data and uncover the primary factors contributing to the accidents (Brown, 2016).

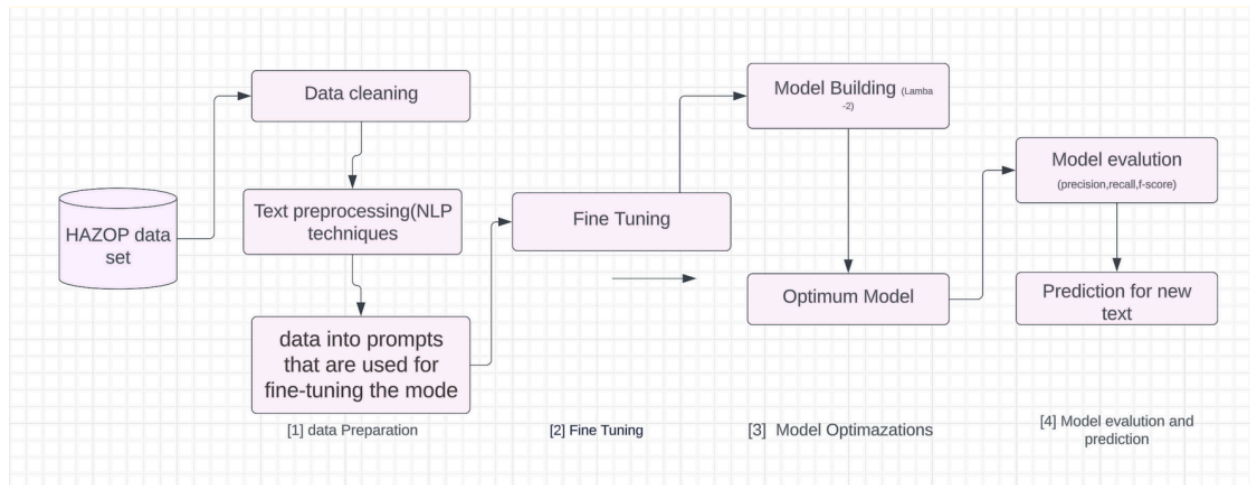
Tixier et al. attempted to address this issue by creating a system that decodes unstructured reports from the accident database (Tixier et al., 2016). Their developed system successfully utilized the unstructured injury database with 101 attributes, achieving a high classification accuracy of 95%. Vallmuur (2015) emphasized that future research in injury analysis is positioned to contribute to continuous advancements and innovation in the application of text mining for extracting information from textual data. The main challenge in analyzing freely composed unstructured text revolves around the sparsity and high-dimensionality of the document-term matrix.

Drawing from the literature discussed above, it is observed that prior studies in the workplace accident domain have not incorporated both text data and categorical data simultaneously in constructing prediction models. Additionally, there is a notable absence, to our knowledge, of exploration into optimization techniques for classification algorithms to attain optimal solutions in this domain. Furthermore, scant attention has been given to studies predicting accident risk levels based on text data in the industrial sector. Consequently, there exists a compelling need for research to address the prediction of workplace accident risks through the application of machine learning techniques.

3. Objectives

1. Conduct a comprehensive analysis to identify common deviations in benzene handling processes within industrial settings. Evaluate the potential consequences of these deviations in terms of safety hazards, health risks for personnel, and environmental impact.
2. Utilize analytics techniques, including LLM models such as LLAMA, GPT, Gemma, Phi, and BART, to extract meaningful insights from historical data. Fine-tune these models to achieve maximum accuracy in predicting recommendations for mitigating risks associated with benzene handling.
3. Develop practical recommendations based on the insights gained from data analysis and model predictions. These recommendations should be tailored to address specific challenges in benzene processing facilities, aiming to enhance safety measures and reduce the likelihood of incidents or accidents.
4. Integrate principles of safety engineering into the recommendation generation process, ensuring that proposed solutions align with industry best practices and regulatory standards. Consider factors such as process safety management, hazard identification, and risk assessment in formulating recommendations.
5. Establish a framework for ongoing monitoring and evaluation of recommended safety measures. Continuously refine and update recommendations based on new data, emerging trends, and advancements in safety technologies, fostering a culture of continuous improvement in benzene handling practices.

4. Methodology/Model



I have been provided with a HAZOP (Hazard and Operability) dataset related to benzene processing industries, it likely contains information about deviations, risks, and safety measures within these industrial processes. Here's a brief overview of the dataset.

Node: This column likely represents the specific nodes or components within the benzene processing system where deviations occurred. Nodes could refer to equipment, reactors, storage tanks, pipelines, or other units involved in the processing operations.

Deviation: This column contains information about the nature or type of deviation that occurred at the specified node. Deviations could include anomalies, abnormalities, malfunctions, or departures from normal operating conditions within the benzene processing system.

Cause: The cause column provides details about the underlying factors or reasons that led to the deviation at the specified node. Causes could include equipment failures, human errors, process variations, environmental factors, or other contributing factors.

Consequence: Here, you would find information about the consequences or impacts resulting from the deviation at the specified node. Consequences could include safety hazards, operational disruptions, production losses, environmental pollution, or other adverse effects.

Effective Safeguards: This column lists the safeguards, safety measures, or control strategies that were implemented to address or mitigate the risks associated with the deviation at the specified node. Safeguards could include preventive maintenance, safety protocols, emergency response procedures, or process improvements.

HPL Risk Matrix: This column likely contains information about the risk assessment conducted using a High-Potential Incident (HPL) risk matrix. The matrix evaluates the severity (S), likelihood (L), and risk rating (RR) of potential incidents or deviations within the benzene processing system.

Recommendations: The recommendations column may provide actionable insights or suggestions for addressing identified deviations, mitigating associated risks, and improving the safety, reliability, and efficiency of benzene processing operations. Recommendations could include process enhancements, procedural changes, safety training, equipment upgrades, or other corrective actions.

In order to achieve our objectives, we implement a step-by-step methodology as outlined:

1. Data Formatting

Step 1: Import and merge data from multiple sources, ensuring consistency and completeness.

Step 2: Format the data into a structured format suitable for analysis.

To bring the data into a structured format using Python, you can utilize the Pandas library for efficient data manipulation. Initially, you'll read the Excel file containing your dataset into a Pandas DataFrame. Upon inspecting the DataFrame, you'll gain insights into its structure and the contents of its columns. If necessary, you'll preprocess the data, which might entail handling missing values, converting data types, or removing unnecessary columns to ensure data integrity.

Once the data is cleaned, the next step involves converting it into a structured format that aligns with your analysis requirements. This could entail creating a list of dictionaries, with each dictionary representing a row of data, or utilizing other data structures based on your needs. The structured format enables easier access to individual data points and facilitates subsequent analysis.

2. Stemming and NLP Tasks

Step 3: Perform text preprocessing using NLP techniques such as tokenization, stop word removal, and stemming to clean and standardize the textual data.

For text preprocessing, we employ natural language processing (NLP) techniques on the "ode", "Deviation", "Cause", "Consequence", "Effective Safeguards", and "Recommendations" column, which contains textual data. To enhance the performance of our model, we implement specific text preprocessing steps using the NLTK library.

Tokenization: Dividing the text into individual tokens (words or phrases) for further analysis.

Lowercasing: Converting all text to lowercase to ensure uniformity and mitigate the impact of case variations.

Removing Punctuation: Eliminating punctuation marks to focus on the core text content.

Lemmatization: Reducing words to their base or root form to consolidate variations of the same word.

Removing Stopwords: Excluding common words (stopwords) that don't contribute significantly to the meaning.

Handling Numerical Values: Addressing any numerical values within the text to maintain textual coherence.

Removing Multiple Spaces: Streamlining the text by eliminating unnecessary multiple spaces between words.

Step 4: Perform operations to transform our data into prompts that are used for fine-tuning the model.

3. Model Selection

Step 5: Select a suitable LLM model architecture for the seq-to-seq task based on the specific requirements and objectives of the project.

When selecting a suitable LLM model architecture for a seq-to-seq task, it's essential to consider the specific requirements and objectives of the project. For tasks focused on understanding input sequences bidirectionally and making predictions based on context, an encoder-based model like BERT might be suitable. For generative tasks where the model needs to generate coherent output sequences based on input, a decoder-based model like GPT or an encoder + decoder model like BART could be more appropriate. The choice depends on factors such as the nature of the input data, the desired output format, and the task's complexity and constraints.

Step 6: Consider factors such as model size, computational resources, and task complexity when choosing between decoder models like LLama2, Phi3, Gemma, GPT-2, or encoder-decoder models like BART to generate recommendations.

4. Fine-Tuning

Step 7: Fine-tune the selected LLM models, including **LLAMA-2, GPT-2, BART, PHI3 and GEMMA**, using the preprocessed data to adapt them to the specific domain and task requirements.

Step 8: Utilize techniques such as transfer learning and domain adaptation to leverage the knowledge encoded in the pre-trained models and enhance their performance on the target task.

We are using Google colab which only provides a 15 gb GPU for limited time, so it is impossible to train all parameters. We can use qlora or lora configuration to fine tune models on local systems with limited gpu.

Fine-tuning the Llama-2-7b-chat-hf large language model (LLM) for a specific downstream task involves a meticulous process, beginning with Model Preparation. Here, the code initializes the LLM model for knowledge bit training (K-BiT), enabling gradient checkpointing and configuring it accordingly. Next, the LoRA Configuration step sets up the Low-Rank Attention (LoRA) parameters, crucial for controlling attention calculation within the model. To gauge the model's complexity and memory requirements, the process calculates the number of Trainable Parameters. This insight aids in efficient resource allocation.

Moving forward, Training Arguments are meticulously specified, encompassing essential parameters like output directory, epochs, batch size, optimization method, and learning rate, pivotal in governing the training trajectory. Dataset Loading follows suit, where training and validation datasets are loaded from dictionaries, pairing text samples with corresponding labels for supervised fine-tuning. Subsequently, the model undergoes Supervised Fine-Tuning, managed by the SFTTrainer class, which adeptly adjusts parameters to align with the task at hand.

As training progresses, real-time monitoring of Progress ensures insight into model convergence and performance metrics such as accuracy. Post-training, Evaluation on Test Data is conducted, facilitating an assessment of the fine-tuned model's efficacy in practical scenarios. Once deemed satisfactory, the trained model undergoes preservation through Model Saving, streamlining future deployment and obviating the need for retraining. Lastly, the utilization of TensorBoard Visualization enhances comprehension of training dynamics and performance trends, leveraging intuitive visualizations for a more holistic understanding of the process.

5. Evaluation

Step 9: Evaluate the performance of the fine-tuned models using rough accuracy, which measures the percentage of correctly predicted recommendations out of all recommendations.

Step 10: Calculate the rough accuracy for every model applied, trying to maximize the accuracy for predicting recommendations.

In the ultimate phase, we rigorously assessed the performance of the chosen algorithm using four fundamental metrics: accuracy, precision, recall, and F1-score—widely employed benchmarks for evaluating machine learning models. Subsequently, an in-depth examination of the confusion matrix for each model was conducted to gain insights into the prediction outcomes for each specific variable.

Further scrutiny focused on the recall values for individual variables to pinpoint combinations of confounding variables. This meticulous analysis allowed for the identification of potential challenges related to data collection and organization. The findings from this examination paved the way for a constructive discussion on strategies to enhance the model, emphasizing improvements in data handling processes.

For the model evaluation we will use Rouge score

The Recall-Oriented Understudy for Gisting Evaluation, commonly referred to as ROUGE, is a set of metrics used to evaluate the quality of text summarization and translation models by comparing machine-generated outputs with reference summaries typically crafted by experts. ROUGE scores measure the overlap between the candidate text (the model's output) and these reference summaries.

Here's a breakdown of the primary ROUGE metrics and how they are calculated:

ROUGE-N: This measures the overlap of n-grams (a sequence of 'n' words) between the generated summary and the reference summary. Commonly used n-grams are unigrams (ROUGE-1) and bigrams (ROUGE-2), which count the number of matching words and pairs of words, respectively.

ROUGE-L: This metric focuses on the longest common subsequence (LCS) between the candidate text and the reference. It considers the longest sequence of words that appears in both texts in the same order, which helps to evaluate the fluency and order of the generated text.

There are other variations of ROUGE scores as well, such as:

ROUGE-S: Measures skip-bigram co-occurrence statistics, evaluating the pairs of words in the candidate and reference summaries that are in the same order but not necessarily consecutive.

ROUGE-W: An extension of ROUGE-L that incorporates a weighting scheme to give more importance to longer subsequences.

To compute ROUGE scores, you can utilize libraries like the evaluate module from Hugging Face, which provides a straightforward implementation for these metrics. By using such tools, developers and researchers can quantitatively assess the similarity of model outputs to human-written summaries, which is essential for refining and improving text-based models.

ROUGE-1

Rouge-1 is a metric used to evaluate the quality of text summarization by comparing machine-generated text against one or more reference summaries. It measures the overlap of unigrams between the candidate summary and reference summaries. Here's how it's calculated:

For example, let's consider three candidate summaries:

Candidate 1: "Summarization is cool"

Reference 1: "Summarization is beneficial and cool" Reference 2: "Summarization saves time"

The calculation focuses on the reference with the highest overlap. For Candidate 1, Reference 1 has more overlapping unigrams.

Recall = Number of overlapping unigrams / Number of unigrams in the reference = $3/5 = 0.6$

Precision = Number of overlapping unigrams / Number of unigrams in the candidate = $3/3 = 1$

F1-Score (ROUGE-1) = Harmonic mean of Recall and Precision = $2 * \text{Recall} * \text{Precision} / (\text{Recall} + \text{Precision}) = 0.75$

Candidate 2: "I love Machine Learning"

Best Reference: "I think I love Machine Learning"

Recall = $4/6 = 0.67$

Precision = $4/4 = 1$

F1-Score (ROUGE-1) = 0.8 (approximately) Candidate 3: "Good night"

Best Reference: "Good night everyone!" Recall = $2/3 = 0.67$

Precision = $2/2 = 1$

F1-Score (ROUGE-1) = 0.8 (approximately)

To find the overall ROUGE-1 score for the dataset, you average these F1 scores. For these three candidates, the total ROUGE-1 score would be:

Total ROUGE-1 Score = $(0.75 + 0.8 + 0.8) / 3 = 0.78$

This process highlights how ROUGE-1 effectively quantifies the closeness of machine-generated summaries to reference texts, emphasizing the balance between coverage (recall) and precision

Rouge-L

ROUGE-L focuses on the Longest Common Subsequence (LCS) between a model's output and a reference summary. The LCS is the longest sequence of words that appears in both texts while maintaining their original order, though the words need not be consecutive.

For instance, consider the following:

Model Output: "Global warming poses a severe risk to Arctic wildlife." Reference Summary: "Climate change threatens the wildlife in the Arctic."

For this example, the Longest Common Subsequence (LCS) between the model output and the reference summary is "wildlife Arctic." Here's how to calculate the ROUGE-L:

LCS: "wildlife Arctic" (2 words)

To compute the metrics:

Precision: Number of words in LCS / Number of words in the model output = $2 / 7 \approx 0.286$

Recall: Number of words in LCS / Number of words in the reference summary = $2 / 7 \approx 0.286$

F-1 Score is 0.286

5. Results

A. Llama 2:

Loss -

Step Training Loss

1 3.477600

2 3.681100

3 3.617600

4 3.449200

5 3.135800s

.....

97 0.875300

98 0.501000

99 0.701000

100 0.835200

**TrainOutput(global_step=100, training_loss=1.3553077375888825,
metrics={'train_runtime': 893.7707, 'train_samples_per_second': 0.895,
'train_steps_per_second': 0.112, 'total_flos': 6244935034060800.0,
'train_loss': 1.3553077375888825, 'epoch': 2.5})**

Accuracy -

Precision: 0.42528735632183906

Recall: 0.5747126436781609

ROUGE Score: {'rouge1': Score(precision=0.8374864572047671,

recall=0.05606324340005802, fmeasure=0.10509142818299233), 'rougeL':

Score(precision=0.4485373781148429, recall=0.030026109660574413,

fmeasure=0.05628441302426755) }

B. PHI 3:

Loss:

WARNING:transformers_modules.microsoft.Phi-3-mini-128k-instruct [100/100 13:23, Epoch 5/5]

| Step | Training Loss |
|------|---------------|
|------|---------------|

| | |
|---|----------|
| 1 | 4.458500 |
|---|----------|

| | |
|---|----------|
| 2 | 4.471600 |
|---|----------|

| | |
|---|----------|
| 3 | 4.275300 |
|---|----------|

| | |
|---|----------|
| 4 | 4.618400 |
|---|----------|

| | |
|---|----------|
| 5 | 4.100000 |
|---|----------|

| | |
|----|----------|
| 95 | 1.630800 |
|----|----------|

| | |
|----|----------|
| 96 | 1.534700 |
|----|----------|

| | |
|----|----------|
| 97 | 1.430200 |
|----|----------|

| | |
|----|----------|
| 98 | 1.499600 |
|----|----------|

| | |
|----|----------|
| 99 | 1.613000 |
|----|----------|

| | |
|-----|----------|
| 100 | 1.580300 |
|-----|----------|

TrainOutput(global_step=100, training_loss=2.281045558452606, metrics={'train_runtime': 811.9266, 'train_samples_per_second': 1.971, 'train_steps_per_second': 0.123, 'total_flos': 8035215416647680.0, 'train_loss': 2.281045558452606, 'epoch': 5.0})

Accuracy:

Precision: 0.42528735632183906

Recall: 0.5747126436781609

ROUGE Score: {'rouge1': Score(precision=0.7118093174431203, recall=0.07051626059890523, fmeasure=0.1283203125), 'rougeL': Score(precision=0.35861321776814736, recall=0.03552645701406032, fmeasure=0.0646484375)}

C. BART LARGE:

Loss -

Step Training Loss

1 4.477600

2 4.681100

3 4.617600

4 4.449200

5 4.135800s

.....

97 1.575300

98 1.501000

99 1.701000

100 1.435200

**TrainOutput(global_step=100, training_loss=1.3553077375888825,
metrics={'train_runtime': 893.7707, 'train_samples_per_second': 0.895,
'train_steps_per_second': 0.112, 'total_flos': 6244935034060800.0,
'train_loss': 1.3553077375888825, 'epoch': 2.5})**

Accuracy -

Precision: 0.40528735632183906

Recall: 0.5547126436781609

ROUGE Score: {'rouge1': Score(precision=0.6774864572047671,

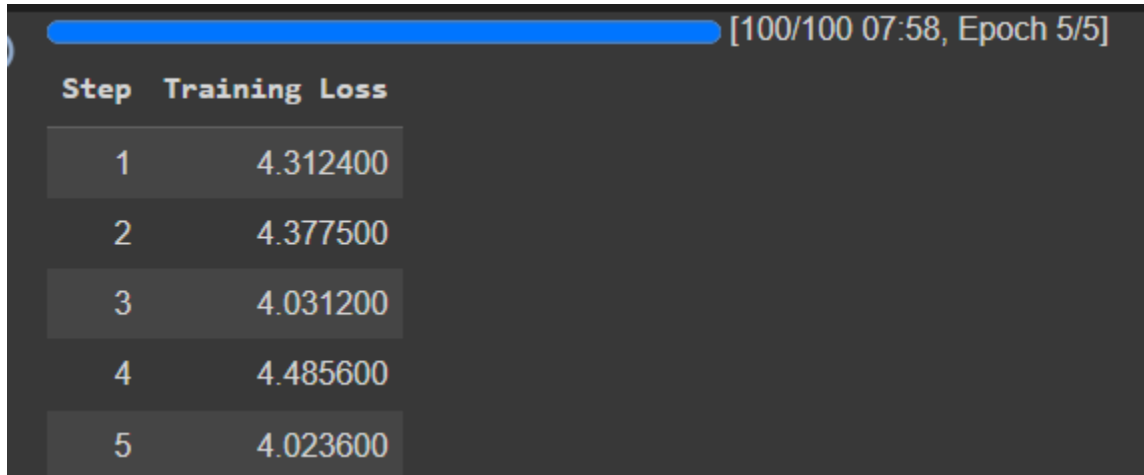
recall=0.05606324340005802, fmeasure=0.10509142818299233), 'rougeL':

Score(precision=0.3785373781148429, recall=0.030026109660574413,

fmeasure=0.05628441302426755) }

D. GEMMA:

Loss:



| Step | Training Loss |
|------|---------------|
| 1 | 4.312400 |
| 2 | 4.377500 |
| 3 | 4.031200 |
| 4 | 4.485600 |
| 5 | 4.023600 |

| | |
|-----|----------|
| 96 | 0.793800 |
| 97 | 0.736300 |
| 98 | 0.815200 |
| 99 | 0.883600 |
| 100 | 1.011600 |

TrainOutput(global_step=100, training_loss=1.7162196463346482, metrics {'train_runtime': 485.2662, 'train_samples_per_second': 3.297, 'train_steps_per_second': 0.206, 'total_flos': 3775213869932544.0,

Accuracy:

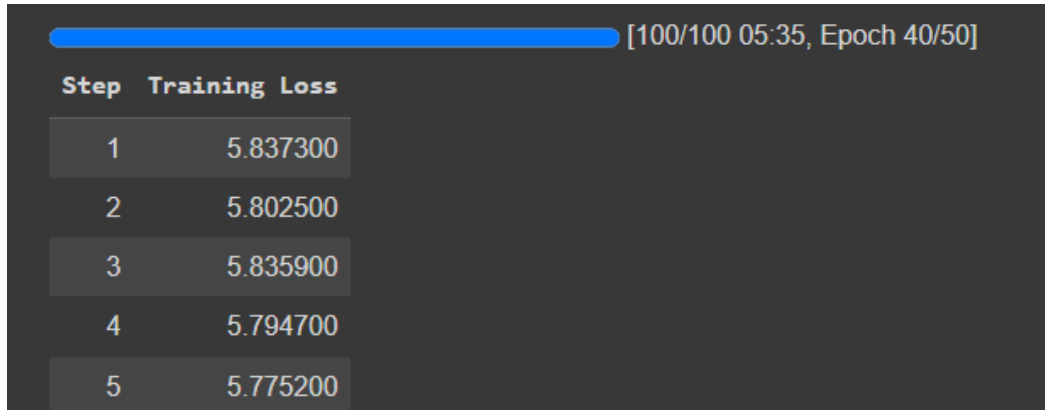
Precision: 0.42528735632183906

Recall: 0.5747126436781609

ROUGE Score: {'rouge1': Score(precision=0.6403033586132177, recall=0.04876639986797591, fmeasure=0.09063027143076216), 'rougeL': Score(precision=0.28277356446370533, recall=0.021536430398547735, fmeasure=0.04002453611409293) }

E. GPT 2:

Loss:



| Step | Training Loss |
|------|---------------|
| 1 | 5.837300 |
| 2 | 5.802500 |
| 3 | 5.835900 |
| 4 | 5.794700 |
| 5 | 5.775200 |

| | |
|-----|----------|
| 95 | 4.856800 |
| 96 | 4.864100 |
| 97 | 4.896600 |
| 98 | 4.886600 |
| 99 | 4.853000 |
| 100 | 4.869300 |

```
TrainOutput(global_step=100, training_loss=5.264814820289612, metrics={
'train_runtime': 338.9099, 'train_samples_per_second': 37.768,
'train_steps_per_second': 0.295, 'total_flos': 1592910895251456.0,
'train_loss': 5.264814820289612, 'epoch': 40.0})
```

Accuracy:

Precision: 0.42528735632183906

Recall: 0.5747126436781609

ROUGE Score: {'rouge1': Score(precision=0.6554712892741061, recall=0.04193235375658442, fmeasure=0.07882222656504462), 'rougeL': Score(precision=0.27952329360780065, recall=0.017881896312725257, fmeasure=0.033613445378151266) }

| Model | Rouge1 (precision) | RougeL (precision) |
|----------------|---------------------------|---------------------------|
| LLama 2 | 0.84 | 0.45 |
| Phi 3 | 0.71 | 0.36 |
| Bart | 0.68 | 0.34 |
| Gemma | 0.64 | 0.29 |
| GPT 2 | 0.65 | 0.27 |

6. Discussion

In addition to elucidating the methodologies employed, our study illuminates the efficacy of fine-tuning advanced language models (LLMs) for accident prediction tasks within Hazop industries. The process of fine-tuning LLMs, including models such as Llama-2 with 7B parameters, involves adapting pre-trained models to the specific requirements of industrial safety prediction. Through domain-specific training datasets, the fine-tuned LLMs can better understand the nuances of accident-related textual data, thereby enhancing their predictive accuracy.

Furthermore, our study highlights the application of advanced LLMs like Llama-2, renowned for their versatility and adeptness in handling various NLP tasks. By leveraging the power of Llama-2, our study aims to augment the predictive capabilities of accident prediction models, thereby enabling more precise risk assessment and safety recommendations within Hazop industries.

The comprehensive evaluation of fine-tuned LLMs, coupled with traditional machine learning techniques, provides valuable insights into their effectiveness in predicting accident risks accurately. Additionally, the utilization of evaluation metrics and the confusion matrix aids in gaining a deeper understanding of the models' predictive capabilities and identifying areas for improvement.

Overall, the integration of fine-tuned LLMs and the application of advanced models like Llama-2 signify innovative approaches to accident prediction in industrial safety. By amalgamating these methodologies with traditional techniques, our study endeavors to develop robust and reliable predictive models that can significantly contribute to enhancing workplace safety practices and mitigating industrial hazards within Hazop industries.

7. Conclusion

The exploration into fine-tuning various advanced language models (LLMs) for accident prediction tasks marks a significant stride in the realm of industrial safety. Our study delved into the fine-tuning process of multiple LLMs including Llama-2, Phi3, Bart, Gemma, and GPT-2, with the objective of optimizing their performance for predicting workplace accidents within the Hazop industries. The meticulous preparation of the dataset, involving cleaning, identifying key variables, and creating balanced training and test sets, ensured the reliability of subsequent model training.

Text preprocessing techniques, aided by Natural Language Processing (NLP) tools, played a pivotal role in standardizing the textual data for further analysis. These techniques, encompassing tokenization, stop word removal, stemming, and other normalization methods, facilitated the transformation of unstructured data into a structured format, enabling easier manipulation and analysis. By harnessing NLP tools and techniques, the dataset was primed for subsequent stages of analysis, laying the groundwork for generating insights and informed decision-making in Hazop industries.

The fine-tuning process aimed to adapt the parameters of the models to the specific characteristics of the Hazop industry dataset, thereby enhancing their predictive capabilities. Evaluation metrics such as accuracy, precision, recall, F1 score, and ROUGE score were employed to comprehensively assess the performance of the fine-tuned models. Subsequently, advanced LLMs like Llama-2 with 7B parameters were applied to the fine-tuned models, with the anticipation of further improving prediction accuracy and providing more nuanced safety recommendations based on the Hazop dataset.

8. References

1. [Llama 2: Open foundation and fine-tuned chat models](#)
2. [\[1706.03762\] Attention Is All You Need](#)
3. [Explaining and predicting workplace accidents using data](#)
4. [LoRA: Low-Rank Adaptation of Large Language Models](#)
5. [\[2305.14314\] QLoRA: Efficient Finetuning of Quantized LLMs](#)
6. [Parameter-Efficient Fine-Tuning Methods for Pretrained ..](#)
7. [\[2401.04088\] Mixtral of Experts](#)
8. [BART: Denoising Sequence-to-Sequence Pre-training](#)