NAUTICAL CRIME INVESTIGATION SERVICES - PROBLEM 3

MASOOMEH AKBARI, MUSTAFA AMIN, NAZANIN HASHEMINEJAD, AND JIALIN HE

Abstract. In this project, we focus on using large language models (LLMs) to extract structured information about sea incidents, a topic of interest for NCIS, from web-scraped content. Our first objective is to generate reports of extracted information from web articles using an LLM. The second objective is to evaluate the performance of different LLMs for the task. To achieve this, we investigated different LLMs and assessed their effectiveness in identifying and extracting relevant information from diverse and potentially noisy datasets. Our findings indicate that employing a combination of LLMs yields optimal performance in retrieving the key information. The approach relies on prompt engineering and optimizing LLMs' performance through careful selection of settings including text splitting, embedding models and the parameter settings of an LLM. Cost minimization was considered throughout the process.

1. Introduction

A large language model (LLM), such as ChatGPT-4, is an advanced AI designed to understand and generate human-like text. These models are trained on extensive datasets and excel in tasks such as answering questions, generating summaries, and extracting key information from text. They leverage deep learning techniques to grasp language nuances, context, and syntax, making them valuable for applications like chatbots, content creation, language translation, and retrieving essential information from large volumes of data.

Nautical Crime Investigation Services (NCIS) aims to play a pivotal role in enhancing national security and maritime defense through advanced technological solutions. The current NCIS project focuses on developing software aimed at identifying, tracking, and building profiles for vessels involved in incidents at sea. This software intends to use LLMs to extract structured information from unstructured articles sourced from the web, particularly data related to sea incidents, and filter and extract key details on entities such as vessel names, flag states, and associated companies from incident reports.

Our objective is to evaluate various LLMs, assess their effectiveness in extracting relevant data, and identify best practices for deploying them.

To achieve our objectives, we started by investigating various LLMs. There exists a multitude of LLMs. For example, Meta AI has introduced the LLaMA series, while Anthropic has launched Claude. Google has contributed with BERT and Gemini, and OpenAI has developed the ChatGPT series. This list, however, is far from comprehensive. Initially, we have identified certain models as the most promising for extracting information. These models include GPT-4o, Gemini 1.5

Pro, LLaMA-3-70B-Instruct, Mistral-Large-2402 and Claude 3 Sonnet. These models are advertised to excel in comprehending and extracting key information from various textual sources. However, the aforementioned LLMs require API keys, incurring costs. Instead, due to time and budget constraints, we decided to proceed with free and locally-run LLMs such as "Aya", "Gemma", "Llama 3", "Mistral", "Phi3", and "Qwen2" for our project. Further, we adopted a certain report format (detailed in Sec. 2) that would allow these generally less powerful LLMs to produce acceptable results, thus showing that the job can be done for a much lower cost.

Having selected several LLMs, the next step involves feeding documents to them. A critical aspect of our approach is prompt engineering. We focus on using clear and detailed prompts to achieve the best results, prioritizing the quality of information received over cost or time optimization.

Alongside prompt engineering, we optimize the LLMs by considering several key factors. Firstly, when feeding a document into the LLM, it's crucial to determine the chunk size and overlap. These parameters directly impact the quality of the model's output by influencing how context is preserved across chunks. Next, incorporating these chunks into a prompt involves embedding them into a vector space that facilitates semantic grouping, stored in a vector store. The choice of embedding model (such as nomic-embed-text, mxbai-embed-large) significantly affects how well semantically similar text can be grouped and retrieved by the LLM. All these variables play a critical role in ensuring the LLM produces a high-quality report.

We use the LangChain framework, which offers a unified workflow for interfacing with multiple LLMs, loading documents, splitting texts, embedding, and storing vectors. We also run LLMs locally.

Finally, we evaluate and compare the results produced by LLMs with those produced by humans for 10 articles. We use various metrics, including span-level Precision, Recall, and F1 Score, to measure the accuracy and completeness of identified entity spans. By evaluating different LLMs and metrics, we identify which models outperform others and pinpoint the most challenging entities to extract by calculating the frequency of entities with the lowest scores.

Name	**Webpage/Provider**	**Price per 1M Tokens**		**Context Window**
		Input	**Output**	
gpt-4o	OpenAI	US$ 5	US$ 15	128K
Gemini 1.5 Pro	Google	US$ 7	US$ 21	1M
llama-3-70b-instruct	Meta	US$ 0.52	US$ 0.75	128k
mistral-large	Mistral	US$ 8	US$ 24	32K
Claude 3 Sonnet	Anthropic	US$ 3	US$ 15	32K
Embed 3	OpenAI	US$ 0.1	US$ 0.1	8K

TABLE 1. Comparison of Token Prices and Context Window for some commercial LLMs

2. Methods

The project has two broad goals: 1. Produce structured reports from unstructured text (articles), 2. Evaluate the produced reports against human-produced reports, and 3. Find the optimal combination of tools to produce a high-quality report.

To produce the output reports, we used the following prompt:

```
Extract information from the retrieved context
context
Your answer must be copied excerpts from the provided text.
You are not allowed to rephrase any part of the provided text.
Your answer for the "Vessel movement" information should be a list of relevant
sentences from the text.
Your answer for the "Entity Identity Information" should be a list of all
mentioned vessels, companies and persons.
Your answer for the "Goods Onboard" information should be a list of all
mentioned goods onboard the vessel.
You must always output only a JSON object with the following keys:
"Offense information",
"Vessel name",
"Vessel flag at the time of the offence",
"Vessel flag at other times if applicable",
"Vessel movement",
"Owner name",
"Owner address",
"Owner country",
"Entity Identity Information",
"Goods Onboard",
"Arrest Information",
"Crew Information and People on Board"
```

2.1. **Our choices and why we made them.**

- The extracted information should be strictly quoted sentences from the input article, without paraphrasing or rewriting. This is for the benefit of both of our broad goals:
  - It will greatly reduce the problem of LLM "hallucination" (the generation of nonsensical text). It also reduces the need for powerful (expensive) models since the problem now is not to "generate" text but rather to search and find the most relevant sentences from the provided article. This allows smaller LLMs to produce good results.
  - It will simplify evaluation of the reports, since now we are not concerned with the LLMs generative capabilities. Further, we gave the human report-writer the same instruction given to the LLMs for consistent scoring.
- We will use free and locally-run LLMs for the following reasons:

  - A naive use of an LLM to extract information from an article requires including the entire article into the prompt. Given that the number of articles to be analyzed could easily be in the tens of thousands, if not more, the cost of using commercial LLMs can be high (see Table 1).
  - The restricted form of the desired output removes the need for a large and powerful LLM.
  - Data security is pivotal for NCIS. Sending articles (including nonpublic ones) along with a descriptive prompts to a third party is potentially risky.
- For running LLMs and embedding models locally, we use Ollama:
  - It is cross-platform and easy to use.
  - It supports many models.
- To interface with LLMs, we will use the python library LangChain:
  - It is popular.
  - It provides a unified interface for all the LLMs we are interested in working with, both commercial and locally-run (through Ollama).
  - It provides multiple useful tools.
  - It has relatively good documentation.

2.2. **Generating the reports.** Since our team did not have the budget to perform tests with commercial LLMs and since we adopted a restricted form of the desired reports, we opted to use freely available and locally-run LLMs for testing. Choosing locally-run tools is also important for We made sure the tools we used apply straightforwardly to commercial LLMs if there is a need.

Under the hood, these are the steps involved in such a process:

(1) The article is subdivided into chunks. This is so that each chunk of text can fit into the "context window" of the LLM. In this step, we have the freedom to choose the **chunk size** and **chunk overlap** (text overlap between chunks). These choices affect the outcome as will be seen in Sec. 5. The values we tested are `500`, `1000` and `3000` characters for chunk size, and `50`, `100` and `250` for chunk overlap
(2) The text of each chunk is "embedded" into a vector store. Text embedding is a process where text is converted into numerical data in a high-dimensional vector space. In that vector space, words the semantically similar will have correspondingly similar vectors. This process is performed by a smaller type of LLM called the embedding model. We have the freedom to choose the **embedding model**. The choice affects the outcome. The models we have tested are `nomic-embed-text` and `mxbai-embed-large`.
(3) Using the embedding vector store, the prompt is evaluated and a "retriever" retrieves the chunks most relevant to the prompt. We have the freedom to choose the vector store. We have not tested the performance of different vector stores and have not measured their effect on the outcome.
(4) The relevant chunk(s), along with the prompt are then passed to an LLM and the output report is produced. The LLMs we tested are: `Aya-8B`, `Gemma-7B`, `Llama3-8B`, `Mistral-7B`, `Phi3-3B` and `Qwen2-7B`. The "nB"

suffix indicates the number of parameters defining the model (number in Billions). These are orders of magnitude fewer than the number of parameters in a commercial LLM. Models of these sizes can run on a machine with 8 GB of (video) RAM.

In running the LLMs, we have the freedom to vary multiple hyperparameters. Among these parameters, we chose to vary the parameter **top_p**, which, in a sense, controls the "creativity" of an LLM in generating text (see arXiv:1904.09751). The parameter takes values only between 0 and 1, which is easier to grasp than the temperature parameter. We tested the values `0.1, 0.2, 0.5` and `0.7`. Initially, we assumed that 0 (or a generally low value) is best, since we require no "creativity" of the LLM to produce our desired reports. Surprisingly, varying this parameter (away from 0) sometimes produces the desired result (see Sec.5).

It should be noted that, even if one desires to use a commercial LLM to produce the reports, steps 1-3 should be done locally so that only chunks of an article are sent to the LLM. This will reduce cost as there will be less tokens to send to the LLM in step 4.

2.3. **Evaluating the reports.** Overall, using all possible combination of the choices discussed above, we generated 432 reports per article. We have tested the setup discussed above for 10 articles. A human writer was given the same prompt as the LLM and produced reports that are considered accurate. A setting combination (chunk size, chunk overlap, embedding model, LLM, top_p) was then scored against the human report as detailed in Sec.3.

2.4. **Future work.** Since we have imposed the restriction that the LLM should provide the extracted information only as excerpts from the original article, this suggests an alternate strategy. Effectively, the problem has now transformed from *generate a text report* to *search for the most relevant sentences in a text*. This suggests that an LLM might not be needed at all for the task, except for the embedding step. We have not explored this strategy.

3. Scoring

In this section, we have already collected the key information extracted through LLMs. Since we already have the information extracted by humans, which could be regarded as the standard correct answer, the next part is to check the accuracy of LLMs.

To check the accuracy of LLMs through comparing the two sets of extracted information, we calculate the similarity of LLM-extracted information against human-extracted information. Therefore, we call this part 'Scoring', which is to use algorithms to give scores to each LLM, after which we will summarize the strategy using these scores.

We divide the 'Scoring' part into three general steps:

3.1. **String Cleaning.** Since the extracted information has a lot of differences, one is the extra symbols, including commas, semicolons, and others like '@#$'.

Another big difference is that sometimes the value type of different sets of information is different; some of them are strings while some of them could be lists and dictionaries. However, as we planned, our algorithms are applied through functions, with two input sentences and one output value as similarity score. So we need to transform lists and dictionaries; here we just simply connect all the elements inside a list or dictionary with commas to form a new sentence.

3.2. **Scoring with Multiple Methods.** We use similarity calculation methods in different types and different focus. Some of them may focus on the shape and distribution of words inside a sentence while some of them might focus on semantics. This part is the main part of the Scoring section and we will introduce the methods below.

3.3. **Storing.** After getting all the similarity scores, before using them to rank LLMs and concluding the strategy, what we need to do first is to store them. Here we constructed a structure of dictionary to store the scores, with the name and setting of the LLM, the article it reads in this term, the entity it wants to extract, and all the scores calculated by different methods.

4. Methods for Scoring

First, we adapted three span-level scores, which are widely used in deep learning validation:

- **Span-level Precision:** Measures the percentage of correctly identified entity spans.
- **Span-level Recall:** Measure the percentage of spans correctly and find all objects of the target entity.
- **Span-level F1 Score:** The harmonic mean of span-level precision and recall.

Then we also use some simple algorithms, just compare the intersection or overlap of two sentences:

- **Intersection over Union (IoU):** IoU can be particularly useful for evaluating the overlap between predicted and ground truth spans, especially when entities are sentences.
- **BLEU score:** It measures the n-gram overlap between the human-extracted sentence and model-extracted sentences, providing a sense of how closely the extracted sentences match the human-annotated ones.
- **Exact Match (EM):** Exact Match measures whether the entire extracted sentence exactly matches the model-extracted sentence.
- **Levenshtein Distance:** Edit Distance measures the minimum number of edits (insertions, deletions, substitutions) required to transform the model-extracted sentence into the human-extracted sentence.

Lastly, we use BERT LLM as an advanced algorithm to calculate the similarity of semantics:

- **WebBERT model:** Loading a BERT LLM first and then using this model to identify the similarity between two sentences. Though it's much slower, it has the highest sensitivity for semantic recognition.

5. Results

As discussed earlier, the aim is to develop a strategy to achieve the highest accuracy in the key information extraction task using an optimal or a combination of optimal LLMs. The models' performance results obtained previously were used to identify the best LLM(s) from various aspects such as:

- General performance
- Performance in retrieving specific key information
- Performance in retrieving the most challenging key information

5.1. **General performance.** Given the limited dataset, we evaluated various LLMs on their ability to extract key information from each article, using a specific scoring metric. We then identified the LLM that consistently outperformed others in accurately retrieving the majority of the required key information.

The results from this limited dataset suggest that Phi-3 outperformed the other models in extracting the majority of key information.

LLM	Company	# of Entities Outperformed
Phi-3	Microsoft	4
Aya	Cohere	3
Llama 3	Meta	2
Gemma	Google	2
Mistral	Mistral AI	1

Table 2. Model Performance Comparison

5.2. **Performance in retrieving specific key information.** For each key information entity, the model demonstrating superior performance, as determined by a pre-selected score, is identified. This superior performance indicates the model's effectiveness in extracting specific data. The goal is to leverage the most efficient models for accurately retrieving each key information entity.

For instance, if model **A** excels in extracting key information **i**, we will use this model specifically for extracting key information **i** from future articles.

5.3. **Performance in retrieving the most challenging key information.** The results were analyzed to identify the key information with the lowest scores for each model. The key information that most frequently exhibited lower scores was identified as the most challenging entity.

The model that outperformed the others in retrieving this challenging entity is identified as a robust model.

Entity	Best_M	Ch_S	Ch_O	Embedding Model	Top_p
Arrest Information	Phi3	500	100	nomic-embed-text	0.7
Crew Information	Phi3	1000	50	mxbai-embed-large	0.5
Entity Identity Information	Llama 3	3000	50	nomic-embed-text	0.5
Goods Onboard	Gemma	3000	250	nomic-embed-text	0.2
Offense Information	Mistral	500	250	nomic-embed-text	0.1
Owner address	Aya	500	250	mxbai-embed-large	0.2
Owner country	Llama 3	3000	250	nomic-embed-text	0.7
Owner name	Phi3	500	100	mxbai-embed-large	0.2
Vessel Name	Phi3	1000	100	mxbai-embed-large	0.1
Vessel flag (at other times)	Aya	3000	50	mxbai-embed-large	0.2
Vessel flag (at offence)	Aya	1000	250	nomic-embed-text	0.1
Vessel movement	Gemma	500	100	mxbai-embed-large	0.2

TABLE 3. Entity Performance by Best LLM

In our case, the most challenging identified entity is "Goods Onboard", with a significant gap compared to the next most challenging entity, which is "Arrest Information". Based on our analysis, Gemma outperformed the other models in extracting "Goods Onboard" from articles.

Entity	Frequency
Goods Onboard	2524
Arrest Information	581
Crew Information and People on Board	509
Owner name	450
Owner country	290
Vessel flag at the time of the offence	152
Vessel movement	130

TABLE 4. Frequency of Lowest Performance by Entity

Entity	LLM
Goods Onboard	Gemma
Arrest Information	Phi3
Crew Information and People on Board	Phi3
Owner name	Phi3
Owner country	Llama 3
Vessel flag at the time of the offence	Aya
Vessel movement	Gemma

TABLE 5. Performance in retrieving the most challenging key information

6. Conclusion and Future Work

We analyzed the performance of several free LLMs, including "Aya", "Gemma", "Llama 3", "Mistral", "Phi3", and "Qwen2" to extract key information from documents related to nautical crimes. Different settings, such as chunk size, chunk overlap, and embedding models, were considered to determine the optimal performance for each model.

Phi-3, developed by Microsoft, demonstrated the best performance overall, successfully identifying four out of seventeen entities, including three of the five most challenging ones. Gemma excelled in extracting the most challenging entity, "Goods on Board", and outperformed other models in retrieving two additional key entities.

Aya, developed by Cohere, excelled in extracting three entities, though none were among the five most challenging. Llama 3, developed by Meta, outperformed in two entities, one of which was among the most challenging. Lastly, Mistral, developed by Mistral AI, successfully extracted one entity, which was not among the most challenging.

Our analysis was limited to these free LLMs and a dataset of only one hundred documents. For more robust and reliable results, a more extensive examination involving a larger dataset and a broader range of LLMs is necessary. Additionally, other factors, such as the exhibition of lower hallucination rates, should be analyzed across different models. It would also be beneficial to develop a strategy, ideally an algorithm, that utilizes these findings to optimize the use of each LLM for key information extraction tasks.

7. Acknowledgement

Striving for improvement embodies our hope for a brighter future, not only for humanity but also for the advancement of AI. We aspire that our work contributes meaningfully to this field. We extend our sincere gratitude to the PIMS Institute for their unwavering support and for providing us the opportunity to apply our knowledge and passion to a real-world challenge. We are particularly grateful to Dr. Kristine Bauer for her support and for organizing the MP2I program. Additionally, we thank the Nautical Crime Investigation Services (NCIS) for presenting us with a compelling problem and for the supervision provided by Sogol Ghattan.