Part 1: Methodology

The first model we used was the fine tuned BERT model. Fine tuning a BERT model for NER[1] involves adapting the pre-trained model to specifics of our task. This allows BERT to leverage the pre-trained contextual understanding for our task. The effectiveness of fine-tuning BERT for Named Entity Recognition (NER) is significantly dependent on the availability of high-quality annotated datasets. In our scenarios where the annotated dataset is limited in size, the model's ability to generalize effectively was slightly compromised. However, Its bidirectional architecture enables it to navigate through intricate language constructs.

Apart from BERT, we also used Zero-shot LLM Baseline and Few-shot LLM Baseline models[2]. In a zero-shot setting, the model performs a task without any task-specific fine-tuning or examples. Instead, it relies on its pre-trained knowledge and prompts designed to guide its behavior. Some of the Pros include that there is no training required. It is directly applicable to a variety of tasks without additional labeled data or fine-tuning. It is also broadly applicable as it leverages the model's general knowledge. However, it has lower task performance because the model lacks task-specific optimization. It is also highly dependent on the design of prompts as variations can lead to different results. This is ideal when there is labeled data or task-specific examples are unavailable.

In a few-shot LLM, the model is provided with a few task-specific examples in the prompt and has no additional fine-tuning. This avoids computationally intensive fine-tuning and there is less reliance on the prompts. The few-shot examples help the model better understand the task**.** However, the performance depends on the quality of the examples.

With these baseline models, we can move onto the experimentation.

The goal of this experiment is to optimize the performance of the LLM for the task of entity labeling. Specifically, we want to improve the F1 score by experimenting with different methods of constructing demonstrations and prompts. Our primary aim is to improve the F1 score by understanding how various aspects of prompt design, such as the number of examples, the structure of examples, and the instructions provided to the model, affect its ability to correctly identify and label entities.

Baseline Setup

The baseline system uses a 5-shot prompt structure for OpenAI's GPT model. This means we include 5 examples from the training set to guide the model in understanding how to label tokens. This baseline performance yields an F1 score of 0.2316602334923164.

Experiment 1

We first experimented with varying the number of shots. Based on the article by __, we can hypothesize that the more examples the model receives, the better it will perform. However, after a certain threshold, there will be diminishing performance[3]. Therefore, we need to find the ideal number of examples (shots)[4].

Research shows that few-shot learning is highly sensitive to the number of examples provided[4]. While 5-shot is a reasonable baseline, we must explore other values.

The different numbers of examples (shots) are: — 1, 3, 5, 10, 20, 30. The model was not able to handle more than 30.

Experiment 2

After finding the ideal number of shots for the baseline model, we can do another experiment. This one has to do with the chat history and the examples provided. We can experiment to figure out what type of prompt engineering will help the model perform better.

According to there are 8 prompt engineering methods: (1) Zero-Shot Learning, (2) One-Shot Learning, (3) Few-Shot Learning, (4) Chain-of-Thought Prompting, (5) Iterative Prompting,(6) Negative Prompting, (7) Hybrid Prompting, and (8) Prompt Chaining[5].

We have already tried the first 3 prompt engineering methods with the first experiment. Therefore, we can try to implement the other 5.

Chain-of-Thought Prompting breaks the problem down into a series of logical steps that the model can follow. For example:

```
You are tasked with labeling entities in the text. The available entity
types are: {', '.join(entity_types_list)}.
In order to label the entities, first label entities based on their
context.
The first mention of an entity gets the 'B-' label (e.g.,
'B-Aquatic_mammal'). The subsequent mentions of the same entity are labeled
with 'I-' (e.g., 'I-Aquatic_mammal'). Non-entities should be labeled as
'O'. Example 1: Text: "The hippopotamus is an aquatic mammal, often seen in
rivers and lakes." Labels: "B-Aquatic_mammal O O O O O O" Example 2: Text:
"Ipet, a goddess from ancient Egyptian mythology, was revered for her
power." Labels: "B-Goddess O O O O O O" """
```

Iterative Prompting builds the response in multiple steps by iteratively refining the model's predictions. However, our API does not allow iterative prompting.

Negative Prompting explicitly instructs the model not to make certain types of predictions or avoid certain biases. Negative prompting might night be the most efficient given our dataset and will not be effective because there is no bias. An example of a negative prompt is:

```
 {'role': 'system', 'content': f""" You are tasked with labeling entities
in the text. Do not label non-entities or make biased and misinformed
predictions. For example, do not label words like 'Ipet' unless it is
clearly referring to the mythological goddess. """}
```

According to Acorn labs, one additional prompting technique that might be useful is multiple choice prompting where we provide multiple possible label choices, and the LLM selects the correct one for each entity[6]. For example:

```
You are tasked with labeling entities in the text. The entity types are:
{', '.join(entity_types_list)}.
For each token, choose the appropriate label from the options below like a
multiple choice question. Example 1: Text: "The hippopotamus is an aquatic
mammal, often seen in rivers and lakes."
Options: - B-Aquatic_mammal - O Labels: B-Aquatic_mammal O O O O O O
Example 2: Text: "Ipet, a goddess from ancient Egyptian mythology, was
revered for her power." Options: - B-Goddess - O Labels: B-Goddess O O O O
O O """}
```

The rest of the methods we mentioned are more involved and are not completely possible with our API.

Research by Wei et al. (2021) has shown that providing structured and unambiguous task instructions helps improve performance in few-shot settings, therefore I hypothesize these will have a higher F1 score[7].

To keep it consistent and to save resources, I tested all with 5 shots and dev_examples_to_take = 10

Part 2: Experimental Results

Experiment 1 Results:

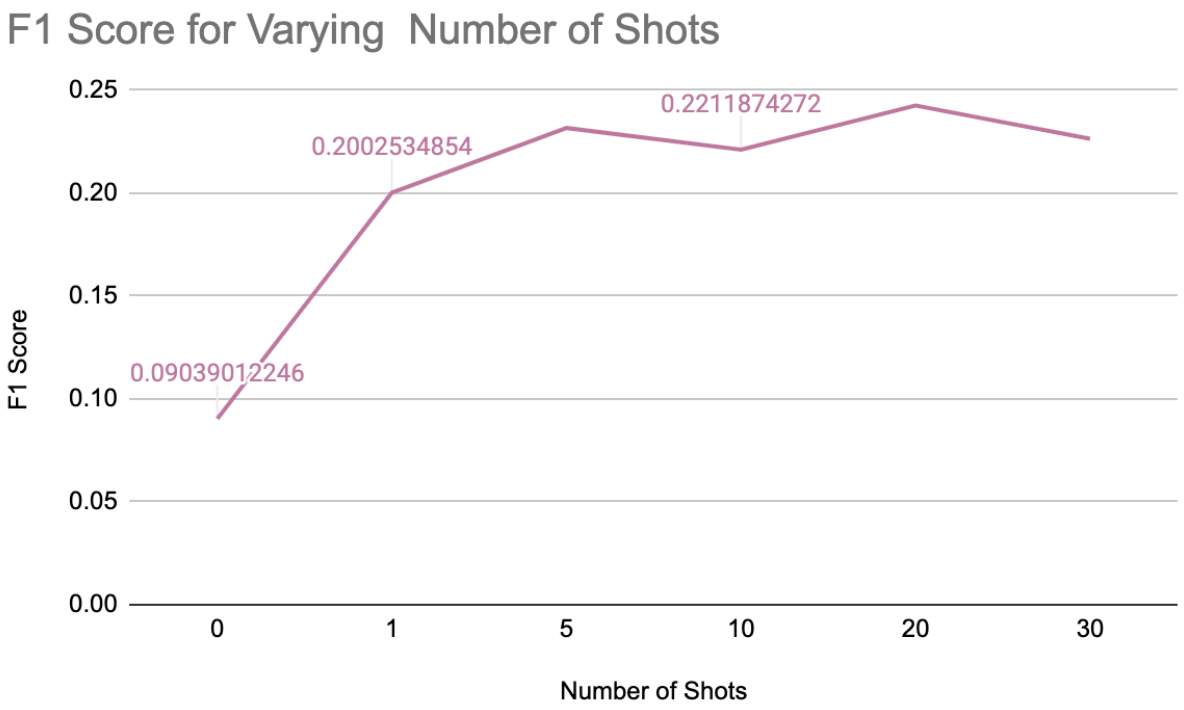
Approach	Shots	Accuracy	Precision	Recall	F1
Finetune BERT	N/A	0.955	0.433	0.479	0.454
Zero-shot LLM	0	0.970048097944906	0.1386211349349111	0.13939393939393938	0.0903901224567721
Few-shot LLM Baseline	1	0.934922023028713	0.1721132897603486	0.23939393939393938	0.2002534854245881
Few-shot LLM Baseline	5	0.9387844337560123	0.2013422818791946 2	0.2727272727272727	0.2316602334923164
Few-shot LLM Baseline	10	0.935505028421513	0.1795841209829867 6	0.2878787878787879	0.2211874272409787
Few-shot LLM Baseline	20	0.946436379536510 7	0.2112359550561797 9	0.2848484848484848 5	0.2425806451612903 3
Few-shot LLM Baseline	30	0.940241947238011 9	0.1951754385964912 2	0.2696969696969697	0.2264631043256997 4

Experiment 2 Results:
(All were tested with 5 shots and 10 examples)

Approach	Shots	Accuracy	Precision	Recall	F1
Few-shot LLM Baseline Baseline	5	0.9270946681175191	0.14285714285714285	0.2962962962962963	0.1927710843373494
Few-shot LLM Baseline Chain of Thought	5	0.9227421109902068	0.22413793103448276	0.48148148148148145	0.30588235294117644
Few-shot LLM Baseline Negative	5	0.9281828073993471	0.0625	0.07407407407407407	0.06779661016949153
Few-shot LLM Baseline Multiple Choice	5	0.9336235038084875	0.26666666666666666	0.4444444444444444	0.3333333333333337

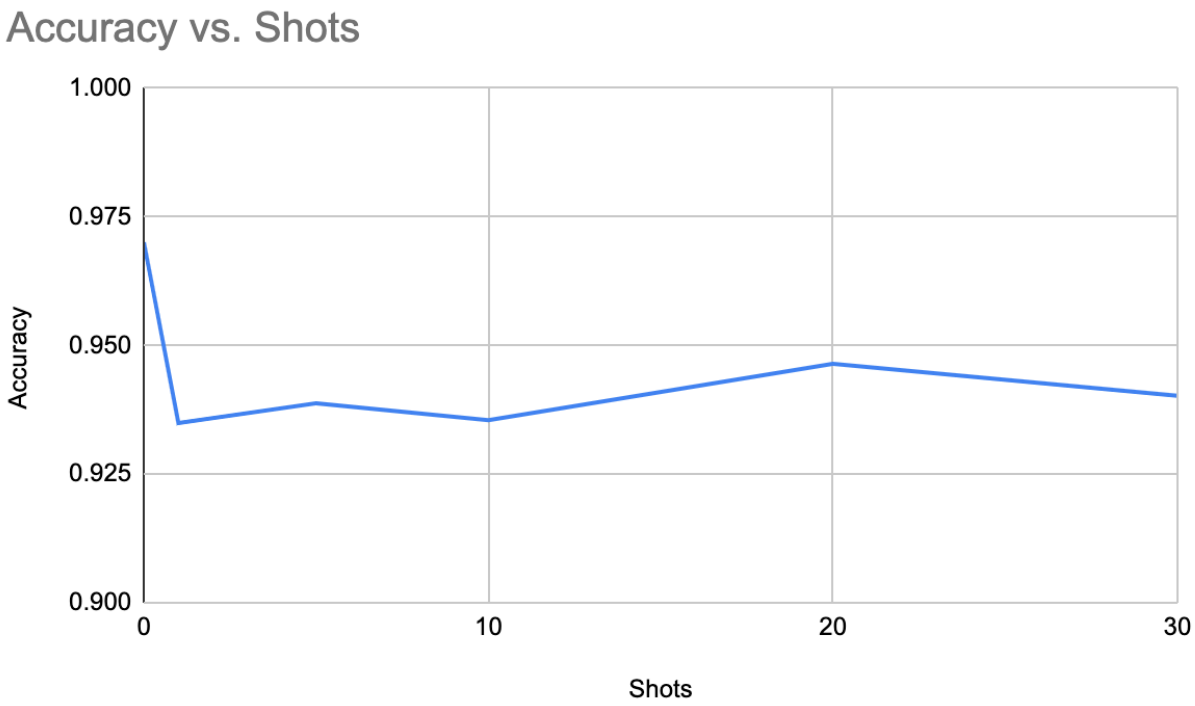
Part 3 Analysis:

From the first experiment, we learned which shot size was the best for our data. After looking at our numerical outputs, these are the results we got:



Based on this graph, we can see that the number of shots that produce the highest F1 score is 20. It has an F1 score of 0.24. The graph also shows the trend of the F1 scores as the number of shots increase. There is a positive correlation until we get to 20 shots and then it decreases from there[2]. According to research, this is a studied concept because there is a threshold and for our model, it was around 20.

Another interesting metric to look at is how the accuracy changes as the number of shots increases.



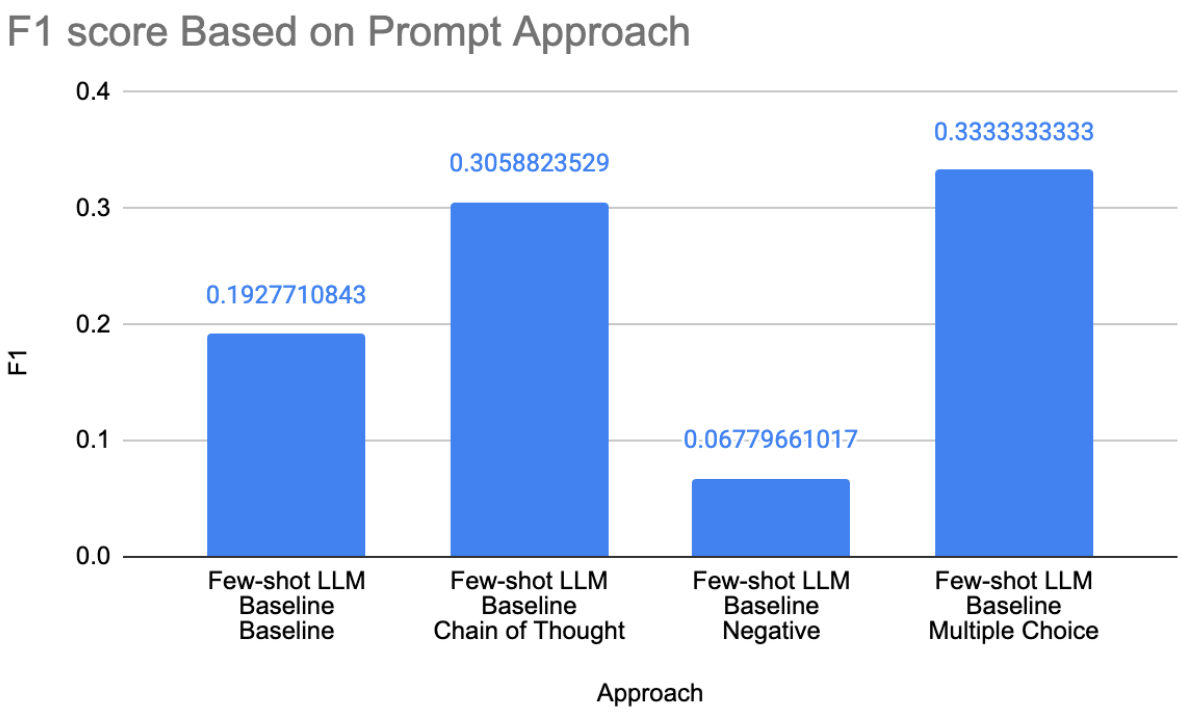
Based on this graph, we can see that the accuracy generally decreases as the number of shots increases[8]. This is because accuracy measures the proportion of correct predictions over all predictions. It does not distinguish between class types. For imbalanced datasets, a model can achieve high accuracy by favoring the majority class while neglecting the minority class. On the other hand, the F1 score is the mean of precision (how many predicted positives are correct) and recall (how many actual positives are identified). It's specifically useful when class distribution is skewed. A model with improved F1 might predict more minority class examples correctly, but if it misclassifies a few more majority class examples, the overall accuracy could drop. Accuracy gives equal weight to all predictions, while F1 emphasizes correct classification of the positive class. If your task prioritizes the minority class, the model might favor increasing F1 at the cost of a slightly lower accuracy.

With my baseline model, I was initially getting a low F1 score. I was not sure why, so I tried changing out the words that were split up and tagged. However, I realized the best way to increase the F1 score was by changing the prompt. Therefore, this was the next part of my experiment – to test different methods of prompts. According to __, prompt engineering is the practice of crafting input queries or instructions to elicit more accurate and desirable outputs from large language models (LLMs). It is a crucial skill for working with artificial intelligence (AI) applications, helping developers achieve better results from language models.Well-designed prompts contribute to model efficiency by reducing unnecessary computation and fine-tuning the model's focus on specific tasks[9].

Therefore, I decided to focus on a few different methods to improve my F1 score. The first method I looked at was Chain-of-Thought Prompting. This is a technique used to improve F1 scores by encouraging the model to generate intermediate reasoning steps when solving complex tasks. The prompt explicitly instructs the model to reason step-by-step or provides examples demonstrating this reasoning. Breaking problems into smaller steps helps the model "remember" intermediate results. However, the model's reasoning abilities are inherently tied to the quality and scope of its training data.

The next method was negative prompting. This is a technique used to guide the model away from generating undesired behaviors. Instead of directly specifying what the model should generate, negative prompting provides explicit instructions or constraints on what the model should avoid. Some problems with this is that it can have excessive constraints that might make the model produce irrelevant or incoherent responses.

The last method I tried was multiple choice prompting. This is a technique used in large language models to solve or reason about tasks involving predefined answer options. In this approach, the model is guided to select or evaluate answers from a list of choices, making it ideal for tasks like quizzes, exams, and structured decision-making. This could lead to low accuracy if the prompt implicitly favors one type of answer format, the model might bias its selections. Therefore, it is best to combine it with chain of thought.



According to this chart created from the data on the different F1 scores, the multiple choice prompting had the highest score. This could be because the model was provided clearer directions.

To conclude, with our data, the LLM model using 20 shots and multiple choice prompting had the highest F1 score.

References:

1. https://medium.com/ubiai-nlp/mastering-named-entity-recognition-with-bert-ca8d04b67b18#:~:text=Fine%2Dtuning%20BERT%20for%20Named,entities%20in%20a%20given%20domain
2. https://medium.com/@sahin.samia/few-shot-and-zero-shot-learning-teaching-ai-with-minimal-data-801603ed40f8
3. https://medium.com/@zbabar/optimizing-large-language-models-using-multi-shot-learning-9ee9eb98709b
4. https://proceedings.neurips.cc/paper/2020/file/1457c0d6bfcb4967418bfb8ac142f64a-Paper.pdf
5. https://medium.com/@amiraryani/8-types-of-prompt-engineering-5322fff77bdf
6. https://www.acorn.io/resources/learning-center/prompt-engineering-examples/
7. https://arxiv.org/pdf/2002.06305
8. https://medium.com/analytics-vidhya/accuracy-vs-f1-score-6258237beca2
9. https://circleci.com/blog/prompt-engineering/#:~:text=Prompt%20engineering%20is%20the%20practice,better%20results%20from%20language%20models

Appendix:

Chain of thought:
Demonstrates how entities are labeled in the provided text using CoT reasoning.
Provides instructions on labeling entities and explains the BIO tagging format with examples.
Example: *"The first mention of an entity gets the 'B-' label, and subsequent mentions get 'I-'"*.

```
def get_chat_history_chain_of_thought(shots, dataset, entity_types_list,
convert_bio_to_prompt_fn):
    chat_history = [
        {'role': 'system', 'content': f"""
You are tasked with labeling entities in the text. The available entity
types are: {', '.join(entity_types_list)}.
In order to label, you must strictly follow these four instructions:
1. First, label entities based on their context.
2. The first mention of an entity gets the 'B-' label. For example,
'B-Aquatic_mammal'.
3. Subsequent mentions of the same entity should be labeled with 'I-'. For
example, 'I-Aquatic_mammal'
4. Finally, Non-entities should be labeled as 'O'.

Here are some examples:
Example 1:
Text: "The hippopotamus is an aquatic mammal, often seen in rivers and
lakes."
Labels: "B-Aquatic_mammal O O O O O O"

Example 2:
Text: "Ipet, a goddess from ancient Egyptian mythology, was revered for her
power."
Labels: "B-Goddess O O O O O O"
"""}
    ]

    for i in range(shots):
        example = dataset[i]
        labeled_text = convert_bio_to_prompt_fn(example)
        chat_history.append({
            'role': 'system',
            'content': f"Example {i+1}:\nText: {'
'.join(example['tokens'])}\nLabels: {labeled_text}"
        })
        chat_history.append({'role': 'user', 'content': f"Text: {'
'.join(example['tokens'])}\nLabels:"})
```

```
    return chat_history
```

Negative:
Focuses on avoiding errors by explicitly stating what not to do (e.g., *"Do not label unrelated or descriptive words"*).
Describes behaviors to avoid, such as over-labeling or bias.

```
def get_chat_history_negative_prompting(shots, dataset, entity_types_list,
convert_bio_to_prompt_fn):
    chat_history = [
        {'role': 'system', 'content': f"""
You are tasked with labeling entities in the text. Do not label
non-entities or make biased and uninformed predictions. For example, do not
label words like 'Ipet' unless it is clearly referring to the mythological
goddess. Avoid over-labeling descriptive and unseen words.
"""}
    ]

    for i in range(shots):
        example = dataset[i]
        labeled_text = convert_bio_to_prompt_fn(example)
        chat_history.append({
            'role': 'system',
            'content': f"Example {i+1}:\nText: {'
'.join(example['tokens'])}\nLabels: {labeled_text}"
        })
        chat_history.append({'role': 'user', 'content': f"Text: {'
'.join(dataset[shots + i]['tokens'])}\nLabels:"})

    return chat_history
```

Multiple Choice:
Guides the model to select labels from a fixed set of options, reducing ambiguity.
Explains how to label entities by selecting the appropriate label from the options.

```
def get_chat_history_multiple_choice(shots, dataset, entity_types_list,
convert_bio_to_prompt_fn):
    chat_history = [
        {'role': 'system', 'content': f"""
You are tasked with labeling entities in the text. The entity types are:
{', '.join(entity_types_list)}.
In order to label the entities first, for each token, choose the
```

```
appropriate label from the options below like multiple choice.

Here are some examples:

Example 1:
Text: "The hippopotamus is an aquatic mammal, often seen in rivers and
lakes."
Options:
- B-Aquatic_mammal
- O
Labels: B-Aquatic_mammal O O O O O O

Example 2:
Text: "Ipet, a goddess from ancient Egyptian mythology, was revered for her
power."
Options:
- B-Goddess
- O
Labels: B-Goddess O O O O O O
"""}
    ]

    for i in range(shots):
        example = dataset[i]
        labeled_text = convert_bio_to_prompt_fn(example)
        chat_history.append({
            'role': 'system',
            'content': f"Example {i+1}:\nText: {'
'.join(example['tokens'])}\nOptions:\n- B-Aquatic_mammal\n- O\nLabels:
{labeled_text}"
        })
        chat_history.append({'role': 'user', 'content': f"Text: {'
'.join(example['tokens'])}\nOptions:\n- B-Aquatic_mammal\n- O\nLabels:"})

    return chat_history
```

Baseline:

Provides general instructions and basic examples without advanced reasoning or constraints.

```
def get_chat_history(shots, dataset, entity_types_list,
convert_bio_to_prompt_fn):
```

```
   chat_history = chat_history = [
   {'role': 'system', 'content': f"""
     This is the list of entity types to label: {',
'.join(entity_types_list)}.
     Label only named entities that are clearly part of the categories
listed.
     For example, 'hippopotamus' is a common noun and should be labeled as
'O' unless explicitly referring to an entity like 'Aquatic_mammal'.
     Similarly, 'Ipet', 'Reret', and 'Hedjet' are only 'Goddess' if
mentioned in mythological contexts, not when they are part of descriptive
phrases.
     Do not label unrelated or ambiguous text.
     Retain punctuation and formatting."""}]

   for i in range(shots):
       example = dataset[i]
       labeled_text = convert_bio_to_prompt_fn(example)
       chat_history.append({
           'role': 'system',
           'content': f"Example {i+1}:\nText: {'
'.join(example['tokens'])}\nLabels: {labeled_text}"
       })
       chat_history.append({
           'role': 'user',
           'content': f"Text: {' '.join(example['tokens'])}\nLabels:"
       })
   return chat_history
```