This project focuses on cross-document event coreference tasks, using a fine-tuned language model and a fine-tuned llm.

Choice of Model: Roberta and Llama-3.2-1B
Fine-tuning techniques: LoRa using PEFT and bitsandbytes

Preprocess:
There are 19649 positive instances and 207679 negative instances in the full training set.
There are 3265 positive instances and 33173 negative instances in the full training set.
There are 3842 positive instances and 39111 negative instances in the full training set.
According to the above data, the proportion of negative and positive instances is obviously unbalanced. Only 8% of the instances are positive in all the datasets. Besides, the training set includes 220k instances, which is too big for fine-tuning an LLM and is going to cost too much time. Thus I decided to downsize the training dataset. The new training dataset will contain all the positive instances and 40k negative instances, with a proportion of 1:2. This will significantly reduce the running time of fine-tuning our Llama model.
The original dataset contains the two target sentences, the index of one trigger word of mentioned event in each sentence, and the label. The trigger word is crucial for the model to learn what event is mentioned in the sentence. I use special tags to enclose the trigger word in each sentence. For example:
The Israeli occupation launched simultaneous <TRG> strikes </TRG> on a number of hospitals during the past hours.
<TRG> and </TRG> indicate the start and end of the trigger word and are added to the original sentence. Two sentences are provided to the tokenizer at the same time as inputs.
Each special token, including those used for highlighting event participants, time, and location, is added to the tokenizer and models additionally. I also assign each special token with a weight (average of all the weights) such that the model can learn something from the special tokens during fine-tuning and can contribute to our task.

Semantic Role Labeling:
Besides the trigger words provided by the original dataset, the participants, location, and time of the event can also provide additional contextual information to our task and potentially enhance model performance. Since this part is not provided by the original dataset, I use Semantic Role Labeling to extract event arguments.
To accomplish SRL, I use allennlp library with *structured-prediction-srl-ber*t model.
I use <ARG> and </ARG> to indicate participant1 and participant2 of the event, <TIME> and </TIME> to indicate the time of the event, and <LOC> </LOC> to indicate the location of the event.
To experiment with SRL features, you can set the parameter srl=True in train.py, or you can extract srl features first and use the pre-processed SRL dataset in the following experiments.

Latter is recommended since producing SRL features costs 4 hours for the downsize training set and 2 hours for the dev and test set.

Baseline: Fine-tuned Roberta, full training set, without SRL features

	Predict Negative	Predict Positive
True Negative	32684	489
True Positive	882	2383

	Negative	Positive
Precision	0.9739	0.8297
Recall	0.9853	0.7299
F1	0.9795	0.7766
Accuracy	0.9623	

The model demonstrates strong performance on negative instances, with high precision (0.9737), recall (0.9853), and F1-score (0.9795), reflecting its ability to correctly identify non-coreferential event mentions. However, its performance on positive instances is weaker, with lower precision (0.8297), recall (0.7299), and F1-score (0.7766). This discrepancy is partially due to the significant class imbalance in the dataset, where negative instances vastly outnumber positive ones. The high overall accuracy (96.23%) is misleading, as it is driven primarily by the model's success on the majority class (negative instances). On the other hand, it is indeed harder for the model to capture the sense of commonality between two sentences, leading to a relatively lower number of correctly identifying coreferential events (positive instances).

Experience 1: Fine-tuned Roberta, downsized training set, with SRL features

	Predict Negative	Predict Positive
True Negative	29294	3879
True Positive	228	3037

	Negative	Positive
Precision	0.9923	0.4391
Recall	0.8830	0.9301
F1	0.9345	0.5966
Accuracy	0.8873	

The introduction of SRL features, including participants, time, and location, has led to mixed results compared to the baseline. While the model's recall for positive instances improved significantly from 0.7299 to 0.9301, indicating better identification of coreferential events, the precision for positive instances dropped sharply from 0.8297 to 0.4391. This suggests that the model is now predicting many more false positives. There are two potential reasons for this decline. First, the additional SRL features may introduce noise or overfitting to specific patterns. Second, the downsized training set might have limited the model's ability to generalize, leading to a lower accuracy. The F1-score for positive instances also decreased from 0.7766 to 0.5966, reflecting the trade-off between precision and recall. On the other hand, the model's performance on negative instances remains strong, with high precision (0.9923) and recall (0.8830), though slightly lower than the baseline. The output suggests that SRL features provide valuable contextual information for identifying coreferential events (positive instances), but we may also need better integration or filtering to reduce false positives.

Fine-tined LLM

I use Roberta and Llama-3.2-1B. Due to my limited computational resources, I could only try a small large language model and use the downsize training set. I choose to use the training set with SRL features because the previous experiment shows that SRL can help identify coreferential events and I want to see if this works for LLM.

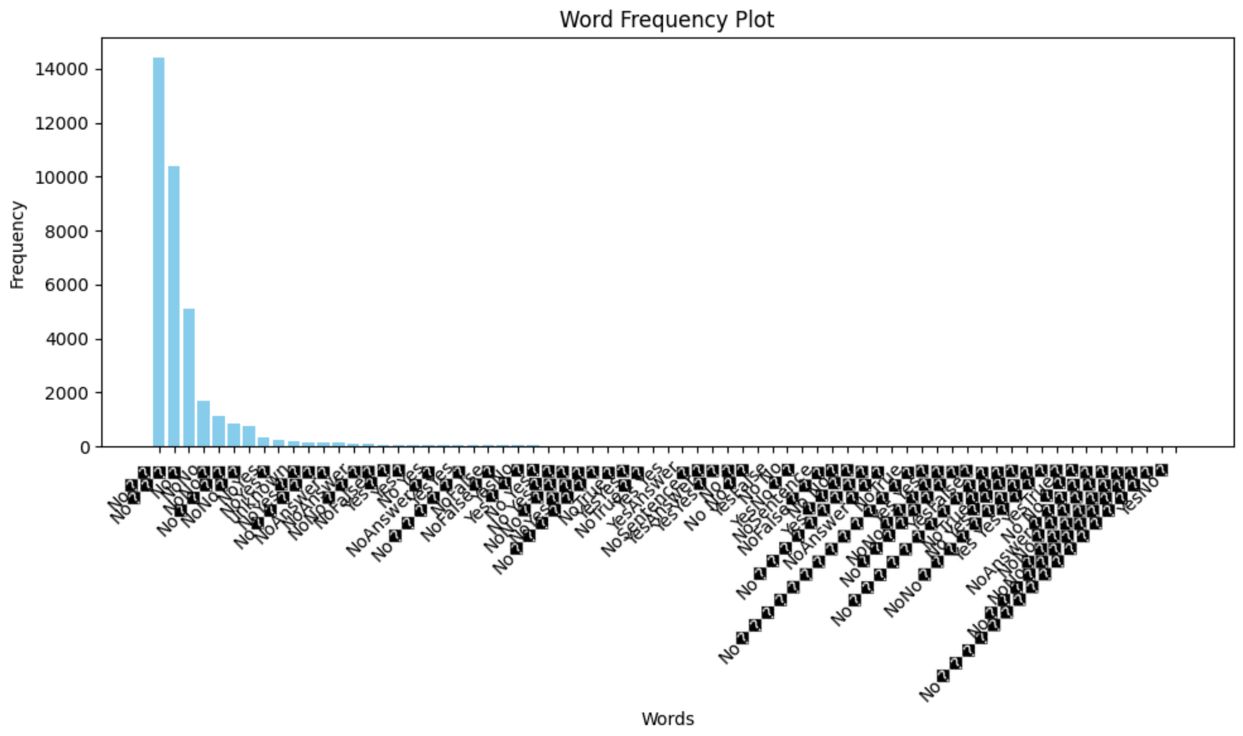
Prompt:

Large Language Models (LLMs) require prompts as input to generate relevant and coherent responses. A prompt serves as a guiding instruction or context, telling the model what kind of output is expected. In the experiment, I tried two different kinds of prompts to test if adding restrictions in the prompt helps regulate the output format. More specifically, the model should return only "yes" and "no" as responses ideally. However, due to the nature of LLM, it generates a response autoregressively, which means it generates text one token (word or subword) at a time, using the previously generated tokens as context to predict the next token. By this, our model can generate any format of response even though we only use "yes" and "no" in training. I experimented with two different prompts in training (lack of computation resources, I have no time to try more.) One prompt specifies the format of output while the other doesn't.

Prompt 1:

```
prompt = (
        f'Question: Does the events mentioned in the following two sentences refer to the same
event?\n'
        f'Sentence 1: {instance["sentence1"]}\n'
        f'Sentence 2: {instance["sentence2"]}\n'
        f'Answer: ' )
```

Output analysis:



As shown in the above picture, the answer is not strictly “Yes” and “No”. I don’t know why most of the answers are in the form of double words (like “No No”). I am also confused about why there are so many Garbled characters in the answers. Since the predicted answers are not well-formed, it is particularly important to set up proper rules to filter “yes” and “no” answers.

Rule 1: [1 if "yes" in word.lower() else 0 for word in responses]

	Predict Negative	Predict Positive
True Negative	32163	1010
True Positive	3127	118

	Negative	Positive
Precision	0.9109	0.1046
Recall	0.9696	0.0361
F1	0.9392	0.0537
Accuracy	0.8859	

Rule 2: [1 if "yes" in word.lower() and “no” not in word.lower else 0 for word in responses]

	Predict Negative	Predict Positive

True Negative	32941	232
True Positive	3232	33

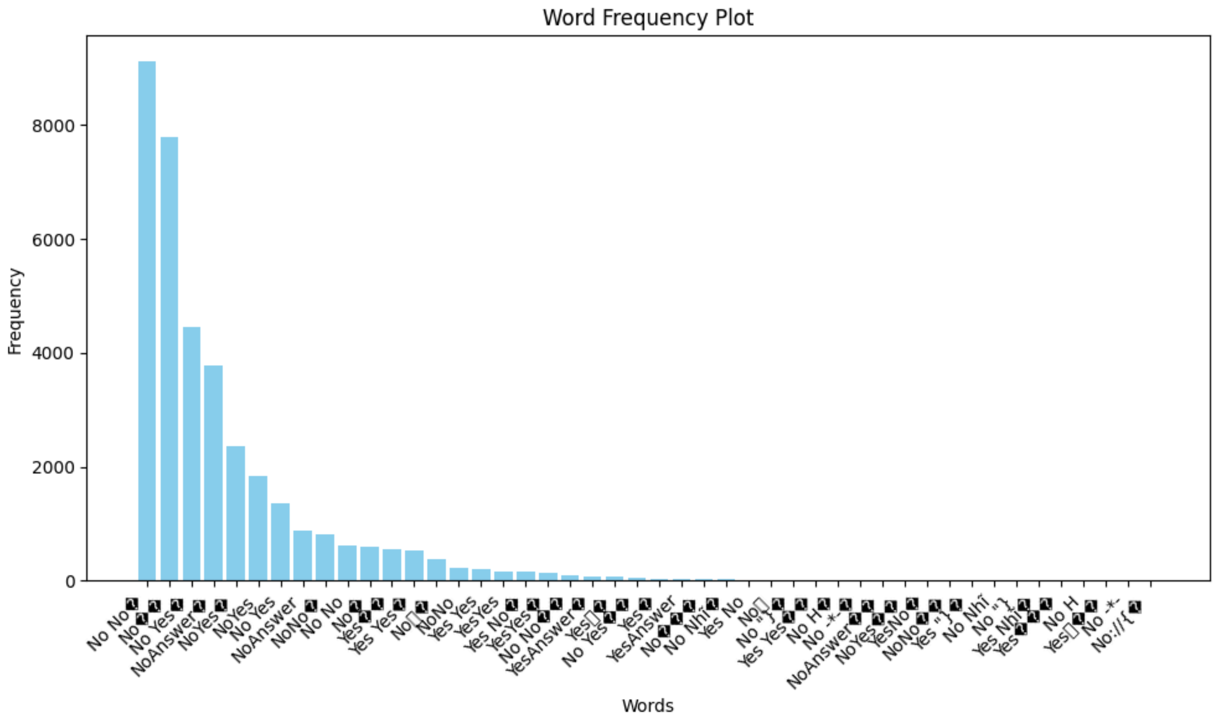
	Negative	Positive
Precision	0.9107	0.1245
Recall	0.9930	0.0101
F1	0.9500	0.0187
Accuracy	0.9049	

In Rule 1, where the model's responses are classified as positive as long as they contain the word "yes" (regardless of whether "no" is also present), the model achieves a high recall for the negative class (0.9696) but a very low recall for the positive class (0.0361), which indicates that the model is biased toward predicting negative outcomes. The low precision and recall for the positive class suggest that the model struggles to generate "yes" responses, even when they are expected. In Rule 2, where the model's responses are classified as positive only if they contain "yes" and do not contain "no," the results show a slight improvement in precision for the positive class (0.1245 compared to 0.1046 in Rule 1). However, the recall for the positive class drops even further (0.0101), indicating that fewer instances are considered positive under stricter criteria of containing "yes" without "no." Overall, the accuracy improves slightly (0.9049 compared to 0.8859 in Rule 1), but this comes at the cost of almost entirely missing positive cases. This suggests that while stricter rules reduce false positives, they also being overly cautious, leading to poor performance in the positive class.

Prompt 2:

```
prompt = (
        f'Question: Does the following two sentences mention the same event? Only answer Yes or No.\n'
        f'Sentence 1: {instance["sentence1"]}\n'
        f'Sentence 2: {instance["sentence2"]}\n'
        f'Answer: ' )
```

Output analysis:



We can notice that with explicit restriction, prompt 2 greatly reduces the number of garbled characters in response. And most of the answers only contain two words. They look way better than the massy output from prompt 1.

Rule 1: [1 if "yes" in word.lower() else 0 for word in responses]

	Predict Negative	Predict Positive
True Negative	22121	11052
True Positive	2242	1023

	Negative	Positive
Precision	0.9080	0.0845
Recall	0.6668	0.3133
F1	0.7689	0.1334
Accuracy	0.6351	

Rule 2: [1 if "yes" in word.lower() and “no” not in word.lower else 0 for word in responses]

	Predict Negative	Predict Positive
True Negative	31484	1698
True Positive	3129	136

	Negative	Positive
Precision	0.9096	0.0745
Recall	0.9490	0.0416
F1	0.9289	0.0534
Accuracy	0.8678	

In Prompt 2, the results for Rule 1 and Rule 2 reveal a trade-off between recall and precision. Under Rule 1, where responses are classified as positive if they contain "yes" (regardless of "no"), the model achieves a higher recall for the positive class (0.3133) compared to Rule 2 (0.0416), but at the cost of lower precision (0.0845). This indicates that although more instances are considered positive, many of them are incorrect, leading to a high number of false positives. The recall for the negative class is also lower (0.6668), suggesting that the model struggles to correctly identify negatives when it is more lenient with positive classifications. In contrast, Rule 2, which requires responses to contain "yes" and exclude "no," significantly improves the recall for the negative class (0.9490) and overall accuracy (0.8678). However, the recall for the positive class drops drastically (0.0416), showing that the stricter rule misses most positive cases. This highlights the challenge of balancing precision and recall when filtering model responses.
When comparing Prompt 2 to Prompt 1, the key difference lies in how the explicit instruction to answer only "Yes" or "No" in Prompt 2 affects the model's behavior. Prompt 2 is more sensitive to strict filer rules, while prompt 1 is more tolerated. The reason may lie in the fact that Prompt 2 uses a restrictive statement in the prompt. Prompt 2 explicitly restricts the answer format in its question. If we allowed considering “No Yes” to be a positive instance, this conflicts with what is pointed out in the question “only answer yes or no”, and consequently largely increases the false positives number. On the other hand, since Prompt 1 doesn’t explicitly mention the answer format, it is more likely to generate a “No Yes” response to the positive instance.
From the above analysis, adding an explicit restriction in the prompt indeed affects the output of LLM. I think prompt 2 is better in the sense of generating quality responses.

Evaluation of Testset:
I picked the "Fine-tuned Roberta, downsized training set, with SRL features" as the best model for fine-tuned language model and "llama with downsized training set, with SRL features, prompt 2, and rule 2" as the fine-tuned LLM. I use them to evaluate the test set, the outputs are as followed:
Fine-tuned Roberta:

	Predict Negative	Predict Positive
True Negative	35139	3967
True Positive	294	3548

	Negative	Positive
Precision	0.9917	0.4721
Recall	0.8986	0.9234
F1	0.9428	0.6248
Accuracy	0.9008	

Fine-tuned Llama:

	Predict Negative	Predict Positive
True Negative	37936	1170
True Positive	3709	133

	Negative	Positive
Precision	0.9109	0.1021
Recall	0.9700	0.0346
F1	0.9396	0.0517
Accuracy	0.8864	

The evaluation results for the Fine-tuned Roberta model demonstrate strong performance across precision, recall, and F1 scores. The model achieves high precision for the negative class (0.9917), indicating that it is highly accurate when predicting negatives. The recall for the negative class is also strong (0.8986), suggesting that the model correctly identifies most true negatives. For the positive class, the model achieves a high recall (0.9234), meaning it captures the majority of true positives, but the precision is lower (0.4721), indicating a higher number of false positives. This trade-off is reflected in the F1 scores, with a high F1 for the negative class (0.9428) and a moderate F1 for the positive class (0.6248). The overall accuracy is 0.9008, showing that the model performs well on the test set, particularly in identifying negatives, but struggles slightly with precision for positives.

In contrast, the Fine-tuned Llama model shows a different performance. It achieves high precision for the negative class (0.9109) and a very high recall (0.9700), indicating that it is highly effective at identifying true negatives and avoiding false negatives. However, the model performs poorly on the positive class, with low precision (0.1021) and extremely low recall (0.0346). This suggests that the model rarely predicts positives and, when it does, many of those predictions are incorrect. The F1 scores reflect this imbalance, with a high F1 for the negative class (0.9396) and a very low F1 for the positive class (0.0517). The overall accuracy is 0.8864, which is slightly lower than the Roberta model, primarily due to the model's inability to correctly identify positive cases.

Conclusion
In conclusion, the small language model like Roberta is better suited for CDEC tasks. I think LLM is not good at this task because of its nature of autoregressive text generation. Llama is more like a text generation model and may generate responses that deviate from the desired format. And since it predicts one token at a time based on previous tokens, it may fail to consider the contextual information that comes from that sentence as a whole. Besides, LLM's performance heavily depends on the prompt design, as seen in the comparison between Prompt 1 and Prompt 2. A bad prompt can lead to noisy or incorrect predictions. This sensitivity makes it less reliable for classification tasks. Roberta, on the other hand, excels at capturing contextual relationships within and across sentences and performs much better in CDEC tasks. Roberta also benefits a lot from SRL features., showing that Roberta can leverage these SRL features which actually contain important information for CDEC while LLM fails to do so.