EventRL: Enhancing Event Extraction with Outcome Supervision for Large Language Models

Anonymous ACL submission

Abstract

This study introduces EventRL, a reinforcement learning approach that significantly enhances the event extraction capabilities of large language models (LLMs). EventRL addresses the challenges of instruction following and hallucination by introducing outcome supervision, which provides direct feedback on the accuracy of event extraction. The method employs specialized reward functions—Argument-F1, Average-F1, and Product-F1—to guide the model's training and improve its understanding of event structures. Our experiments on the ACE05 dataset, which includes both Held-in Test (for seen event types) and Held-out Test (for unseen event types), demonstrate that EventRL outperforms Supervised Fine-Tuning (SFT) and Few-Shot Prompting (FSP) (based on GPT4) methods for event extraction. The results further show that EventRL is particularly effective in handling unseen event types, and that the choice of reward function and the inclusion of code data can significantly improve event extraction performance.

1 Introduction

Event extraction, a crucial task in natural language processing (NLP), aims at identifying and categorizing events within texts (Chen et al., 2015; Nguyen et al., 2016; Liu et al., 2018; Yang et al., 2019; Lu et al., 2021; Gao et al., 2023a). Recently, large language models (LLMs) have demonstrated impressive capabilities in language understanding and generation for various tasks (Ouyang et al., 2022; Chen et al., 2021; Achiam et al., 2023; Zhao et al., 2023). However, they also encounter specific challenges, such as *instruction following* (Zhou et al., 2023; Zeng et al., 2023) and the generation of inaccurate or irrelevant content, often referred to as *hallucinations* (Li et al., 2023).

In the context of event extraction tasks, these models encounter similar difficulties, including mismatches in event structure and the generation of undefined event types (Gao et al., 2023a). As illustrated in Figure 1, a mismatch in event structure refers to inaccuracies like incorrectly including an irrelevant argument. For instance, an "Attack" event erroneously featuring a non-existent "entity" role. The generation of undefined event types refers to the model's prediction of event types that are not predefined in the task instructions. For example, in texts related to elections, the model might unpredictably predict a "Vote" event type, which is not defined in the original guidelines. These issues can be seen as manifestations of instruction following and hallucination problems within the realm of event extraction. Recent studies (Wang et al., 2023a; Gao et al., 2023b; Sainz et al., 2023) have attempted to address these challenges using Supervised Fine-Tuning (SFT) methods, but the performance has been far from satisfactory. A potential

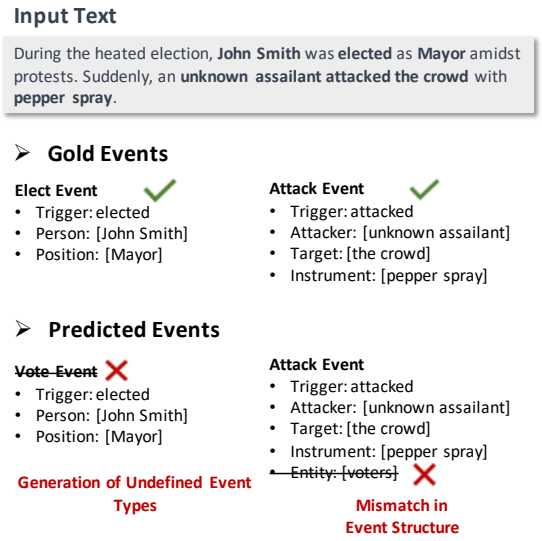


Figure 1: Examples of common errors in LLM-Based event extraction. The left side depicts an error of generating an undefined event type, specifically an unexpected "Vote" event not included in the guidelines. The right side shows a structural mismatch error within an "Attack" event, incorporating an "Entity" argument that deviates from the pre-defined event schema.

reason for this is that event extraction demands too much recognition for abstract concepts and relations, and it suggests that there is a need to focus more on enhancing the high-level understanding for event comprehension (Huang et al., 2023a).

A major limitation of SFT approaches in event extraction is their inability to accurately recognize errors in event structures, such as incorrect argument inclusion or predicting events not defined in the guidelines. This issue may stem from the reliance on Negative Log Likelihood (NLL) loss, which, while effective for general language modeling, falls short in capturing the intricacies of event extraction. Specifically, when it comes to event extraction, both types of errors—incorrect predictions of event types and incorrect predictions of an event argument's role—often differ from the correct samples by just a single word in the text. However, in terms of NLL loss, these errors result in only minor differences, failing to reflect their significant impact on event extraction performance. This discrepancy is particularly critical because an error in predicting the event type can lead to a cascade of errors in the extraction of all associated arguments, drastically reducing the accuracy of the entire extraction process. Therefore, while NLL loss might marginally penalize these mistakes, their actual consequences are far more severe.

One potential solution is to integrate feedback on the model's performance in identifying and structuring events into its training process. This method, known as outcome supervision, draws inspiration from previous works in solving math problems (Uesato et al., 2022; Lightman et al., 2023). By incorporating outcome-based feedback, the model can adjust and refine its strategies for more accurate event identification and structuring, addressing the specific challenges it faces in understanding and extracting events from text. To this end, we introduce EventRL, a novel approach that uses reinforcement learning to integrate direct feedback on the accuracy of event extraction. Unlike current SFT approaches, EventRL focuses on the specific challenges of LLMs in event extraction, i.e., the mismatches in event structure and the generation of undefined event types. It leverages outcome performance as feedback to penalize errors, guiding the model to adjust its strategies for better performance. We explore three event-specific reward functions: Argument-F1, Average-F1, and Product-F1. These functions are designed to improve the model's understanding of event structures, ensuring more accurate and reliable event extraction. This approach marks a significant step forward in addressing the limitations of current LLMs in event extraction, enhancing their ability to understand and extract events from text accurately. Our contributions can be summarized as follows:

- We introduce outcome supervision to LLMs for event extraction, focusing on task outcomes to improve event understanding and extraction. To the best of our knowledge, we are the first to incorporate outcome feedback on event structures into the training process of LLMs for EE.
- We develop EventRL, a novel approach that implements outcome supervision through reinforcement learning with tailored reward functions, to provide a more precise and targeted training method for EE.
- Extensive experiments with LLMs of varying sizes show that EventRL significantly outperforms standard SFT methods, confirming the benefits of outcome supervision in enhancing LLMs' performance in EE.

2 Related Work

Event Extraction Event Extraction (EE) has evolved from traditional sequence labeling methods to the integration of advanced machine learning models, particularly large language models. The initial approach to EE focused on word-level classification, capturing sentence dependencies (Chen et al., 2015; Nguyen et al., 2016; Liu et al., 2018; Yang et al., 2019; Wadden et al., 2019). A significant shift occurred with the introduction of Machine Reading Comprehension techniques, which transformed EE into a question-answering task, enhancing event extraction (Chen et al., 2020; Du and Cardie, 2020; Li et al., 2020; Zhou et al., 2021; Wei et al., 2021). The subsequent development of sequence-to-structure generation with Transformer-based architectures further streamlined the process by merging event detection and argument extraction (Lu et al., 2021, 2022; Lou et al., 2023). The latest advancement involves LLMs, which, due to their extensive pre-training, demonstrate exceptional generalization capabilities, advancing traditional EE techniques and enabling zero-shot event extraction, marking a notable progression in NLP (Wei et al., 2023; Gao et al., 2023a; Wang et al., 2023a; Sainz et al., 2023; Gao et al., 2023b).

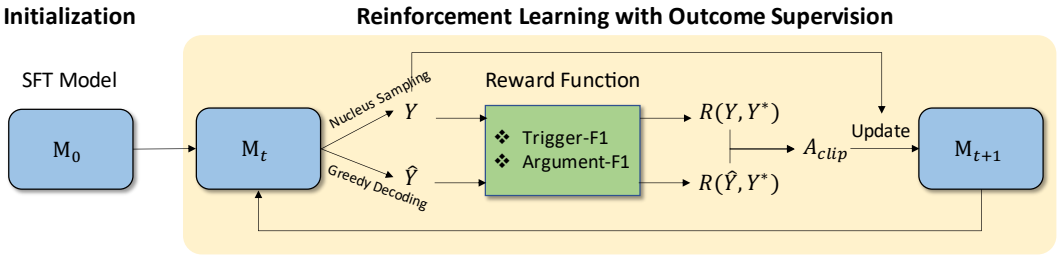


Figure 2: The EventRL framework architecture, demonstrating the process from initialization with an SFT Model M_0, through iterative model updates M_t to M_{t+1} via Outcome Supervision. This includes using reinforcement learning with reward functions based on Trigger-F1 and Argument-F1 scores, which guide policy gradient updates for enhanced event extraction from text.

In contrast to previous work, our work focuses on utilizing outcome supervision to refine model training of LLMs for event extraction, thereby enhancing performance. Notably, we are the first to incorporate outcome feedback into the LLM training process for EE.

Large Language Models and Outcome Supervision Large language models has marked a significant advance in the field of NLP. Recent studies have demonstrated the exceptional capability of LLMs, such as ChatGPT and GPT-4, to perform with remarkable performance in event extraction (Gao et al., 2023a; Sainz et al., 2023). These models demonstrate notable performance gain even in zero-shot learning settings, indicating their potential to generalize across different types of event-related information without the need for task-specific training data. Despite the progress, LLMs continue to face challenges related to instruction following (Ouyang et al., 2022) and hallucinations (Huang et al., 2023b; Zhang et al., 2023b; Li et al., 2024). To address these issues, researchers have explored a range of strategies, including Supervised Fine-Tuning (SFT) (Wang et al., 2022; Zhang et al., 2023a; Wang et al., 2023b), Reinforcement Learning from Human Feedback (RLHF) (Stiennon et al., 2020; Ouyang et al., 2022; Kaufmann et al., 2023), and more recently, approaches that combine outcome supervision with reinforcement learning techniques (Uesato et al., 2022; Lightman et al., 2023; Yu et al., 2023) for solving math problems. Building on these seminal works, our work makes the first attempt to introduce outcome supervision for event extraction tasks, which can harness the power of LLMs while directly address their limitations in instruction following and hallucinations, thus significantly improving the efficacy and reliability of event extraction.

3 EventRL

3.1 Overview

EventRL aims to enhance event extraction through outcome supervision with reinforcement learning for LLMs. As depicted in Figure 2, EventRL begins with an SFT phase to establish a baseline understanding of event extraction. It then progresses to implement outcome supervision, leveraging reward functions based on resulting performance (Trigger-F1 and Argument-F1 scores) to guide the model's training via reinforcement learning techniques. To ensure stable and effective learning, EventRL incorporates stabilization strategies, including a Teacher-Force Threshold and Advantage Clipping, which are critical for mitigating policy degradation and preventing catastrophic forgetting. The subsequent sections will delve into the detailed implementation of these components.

3.2 Initialization

Input and Output Format In the Initialization phase of EventRL, we adopt a hybrid input and output format, building upon the work of Sainz et al. (2023). As illustrated in Figure 3, our approach combines structured Python dataclass formats for event definitions with natural language instructions for task descriptions. This allows for precise event representation while maintaining user-friendly instructions. The output is a Python list of dataclass instances, representing the extracted events in a structured and programmatically accessible format. This hybrid format enhances the model's ability to process and output complex event information accurately, bridging the gap between structured coding and natural language understanding.

Supervised Fine-tuning The SFT process (Wang et al., 2022; Zhang et al., 2023a; Wang et al., 2023b; Peng et al., 2023) in our approach

 # Event Definitions

```python
@dataclass
class Elect(Event):
"""An Elect Event takes place whenever a candidate is victorious in an election
that is intended to establish the Person
role within a StartPosition Event."""

    mention: str # The text span that most clearly expresses (triggers) the event
    person: List[str] # The person elected
    entity: List[str] # The voting agent(s)
    position: List[str] # The JobTitle for the position being nominated to
    time: List[str] # When the election takes place
    place: List[str] # Where the election takes place

@dataclass
class Attack(Event):
    ...
```

 # Instruction

Your task is to analyze the given text and extract any relevant events based on the provided event data structures. After analyzing the text, you should output the results as a Python list of dataclass instances, each representing an identified event. Ensure that each instance is properly filled with the corresponding attributes according to the event type. If certain information is not available in the text for any attribute, leave the corresponding list empty.

```python
text = "I didn't particularly like the the tone about the the religious uh -- side of -- the that election ."
```

 # Response

The annotation instances that take place in the text above are listed here:

```python
result = [
    Elect(mention="election", person=[], entity=[], position=[], time=[], place=[])
    ]
```

Figure 3: Illustration of the input-output format in EventRL for event extraction. The input includes Event Definitions in Python dataclass format and a natural language instruction. The output showcases a Python list of dataclass instances as the Response, representing extracted events from the given text. The complete event definitions can be found in Figure 12 in Appendix.

is an important initial phase that establishes a foundational understanding of the event extraction process in the model. During SFT, the model is trained using a labeled dataset that consists of examples, where each event is explicitly defined with its corresponding triggers and arguments. This dataset provides clear and structured examples of desired event extraction outputs.

3.3 Outcome Supervision with RL

Problem Formulation To implement outcome supervision, we leverage reinforcement learning (Kaelbling et al., 1996). The RL method treats event extraction as a sequential decision-making process, where the model M, guided by its policy π_θ, generates predictions Y, which include event triggers and their arguments, based on inputs X.

The model's parameters are updated to maximize expected rewards, leveraging an advantage-based policy optimization method to guide learning (Sutton and Barto, 2018). The update rule is as follows:

$$\Delta\theta \propto \nabla_\theta \log \pi_\theta(Y|X) \cdot A, \tag{1}$$

where $\nabla_\theta \log \pi_\theta(Y|X)$ is the gradient of the log-probability of taking action Y in state X under policy π_θ. The advantage function is $A = R(Y, Y^*) - b$, where $R(Y, Y^*)$ is the reward function and b is a baseline reward. This function calculates the difference between the reward $R(Y, Y^*)$ for the model's predictions and a baseline b, identifying actions that yield above-average benefits. This approach stabilizes policy gradient estimates by reducing variance (Rennie et al., 2017).

Reward Function In our EventRL framework, the reward function focuses on two primary aspects: Trigger Extraction and Argument Extraction. These aspects are quantified through Trigger-F1 and Argument-F1 scores, respectively, which serve as the basis for our reward function. The Trigger-F1 score assesses the model's ability to accurately identify and classify event triggers, while the Argument-F1 score evaluates the precision in identifying and classifying the arguments associated with those triggers. Following previous work (Lu et al., 2021), we adopt the following criteria of the evaluation: A trigger is correct if its event type matches with the ground truth. Similarly, an argument is considered correctly identified if its event type and role match with the ground truth.

In our work, we explore three different reward function designs, shown in the following equation:

$$R = \begin{cases} F1_{\text{arg}}, & \text{Argument-F1} \\ \frac{1}{2}(F1_{\text{trig}} + F1_{\text{arg}}), & \text{Average-F1} \\ F1_{\text{trig}} \times F1_{\text{arg}}, & \text{Product-F1} \end{cases} \tag{2}$$

The Argument-F1 reward, $F1_{\text{arg}}$, focuses on how well the model can identify and classify the arguments of events. Meanwhile, the Average-F1 reward, $\frac{1}{2}(F1_{\text{trig}} + F1_{\text{arg}})$, gives a balanced look by combining Trigger-F1 and Argument-F1 scores, offering a full view of the effectiveness of the model. Lastly, the Product-F1 reward, $F1_{\text{trig}} \times F1_{\text{arg}}$, highlights the importance of doing well in both trigger detection and argument extraction by multiplying these scores together, pushing the model to excel in both areas for a better total reward.

In current approaches of instruction tuning, especially in response generation tasks, feedback often comes from model scoring. This can involve predictions from a reward model trained on human preference data (Ouyang et al., 2022) or direct scoring by more advanced models like GPT-4 (Cui et al., 2023). Although, for event extraction tasks, we could train a reward model on positive and negative examples or have GPT-4 score the outputs,

the necessity for such methods is reduced. This is because standard evaluation metrics for event extraction already provide a clear reflection of output quality. However, there is still value in the model generating natural language judgements (Xu et al., 2023) to further address and correct training issues. We leave this for future work.

Advantage Calculation To calculate the advantage, we compare the rewards of two distinct strategies for extracting events: greedy decoding and nucleus sampling. When processing a given text, the model generates two outputs: one through greedy decoding ($\hat{Y}$) and another via nucleus sampling (Y). We then assess these outputs by calculating their rewards, $R(\hat{Y}, Y^*)$ for the greedy decoding output and $R(Y, Y^*)$ for the nucleus sampling output. The advantage function $A(\cdot)$ quantifies the benefit of the nucleus sampling strategy over the greedy decoding by measuring the difference in their rewards:

$$A = R(Y, Y^*) - R(\hat{Y}, Y^*). \quad (3)$$

This comparison highlights the effectiveness of exploratory actions in improving event extraction outcomes compared to the baseline approach.

3.4 Stabilization Strategies in EventRL

To ensure stable training, two key stabilization strategies are implemented: the Teacher-Force Threshold (Bengio et al., 2015) and Advantage Clipping (Schulman et al., 2017).

Teacher-Force Threshold The Teacher-Force Threshold strategy is employed to mitigate policy degradation, especially in instances where the model's performance on certain samples is significantly below par. This strategy involves setting a threshold value, denoted as τ, for the model's performance score (typically measured using the outcome score from greedy decoding). When the model's performance on a given sample falls below τ, the learning process is adjusted to "teacher forcing" mode. In this mode, the model is temporarily guided using the gold events Y^* instead of its own generated output Y, effectively providing a more reliable learning signal.

Advantage Clipping Advantage Clipping in the EventRL is specifically designed to address the challenge of catastrophic forgetting, a phenomenon where the model's performance on previously learned tasks deteriorates as it focuses on new ones. This issue often arises when the advantage values for certain samples are too low, leading to negligible updates and causing the model to overlook these samples during training. The strategy involves setting a lower bound for the advantage values, denoted as A_{min}. This lower bound ensures that every sample, regardless of its initial advantage value, contributes a minimum threshold of influence to the learning process. The advantage clipping process is reformulated to focus solely on the lower bound, as follows: $A_{\text{clip}} = \max(A, A_{\text{min}})$.

4 Experimental Setup

4.1 Dataset and Data Splitting Strategy

To evaluate our model's performance comprehensively, we conducted experiments on the ACE05 dataset (Christopher et al.), known for its diversity in event types. This allows us to test the model's capability in extracting both familiar (seen) and novel (unseen) event types effectively. The ACE05 dataset, which contains a total of 33 event types, was used to construct our experimental setup. We selected 7 event types for the training set, validation set, and the held-in test set (seen event types). We then chose 19 different event types to form the held-out test set (unseen event types), ensuring a rigorous evaluation of the model's generalization abilities. To maintain a balanced dataset, we sampled 50 instances for each event type in the training set, 10 for each in the validation set, and 20 for each in both the held-in and held-out test sets. This strategy ensures that the model is trained and evaluated under varied conditions, providing a comprehensive understanding of its performance across different event types. For detailed statistical information, please refer to Appendix A.1.

4.2 Comparison Methods

In our study, we assess the efficacy of EventRL against current methods like Few-Shot Prompting (FSP) and Supervised Fine-Tuning (SFT), applying these methods across various LLMs to evaluate performance in event extraction: **(1) Few-Shot Prompting (FSP):** Implemented on the specific version of **GPT-4** (API version 2023-05-15), provided by the Azure OpenAI Service, this method relies on the model's intrinsic capabilities by providing a limited set of examples before task execution. **(2) Supervised Fine-Tuning (SFT):** This approach involves direct training of models on specific datasets, with feedback provided via NLL

Method	Held-in test			Held-out test		
	Trigger	Argument	AVG	Trigger	Argument	AVG
GPT4 + FSP (0-Shot)	6.04	22.08	14.06	15.42	17.69	16.56
GPT4 + FSP (1-Shot)	23.02	22.82	22.92	19.32	17.83	18.58
GPT4 + FSP (2-Shot)	24.65	23.48	24.06	24.12	18.31	21.22
GPT4 + FSP (3-Shot)	31.58	23.53	27.55	27.07	18.61	22.84
LLaMa-7B + SFT	71.33	40.74	56.03	48.51	26.18	37.35
LLaMa-7B + EventRL (Arg-F1)	73.06	42.34	57.70	51.15	29.32	40.23
LLaMa-7B + EventRL (AVG-F1)	72.34	42.29	57.32	54.59	29.81	42.20
LLaMa-7B + EventRL (Prod-F1)	72.03	49.41	60.72	51.71	29.97	40.84
LLaMa-13B + SFT	76.23	51.16	63.69	51.61	32.46	42.04
LLaMa-13B + EventRL (Arg-F1)	77.61	51.93	64.77	53.07	32.83	42.95
LLaMa-13B + EventRL (AVG-F1)	77.26	54.55	65.90	51.53	34.93	43.23
LLaMa-13B + EventRL (Prod-F1)	76.23	51.66	63.94	53.79	35.03	44.41
CodeLLaMa-7B + SFT	74.31	44.16	59.23	62.21	37.26	49.74
CodeLLaMa-7B + EventRL (Arg-F1)	75.35	50.84	63.09	61.64	37.93	49.78
CodeLLaMa-7B + EventRL (AVG-F1)	77.14	47.06	62.10	61.62	39.01	50.32
CodeLLaMa-7B + EventRL (Prod-F1)	76.60	48.39	62.49	60.34	39.69	50.01
CodeLLaMa-13B + SFT	77.70	47.21	62.46	61.40	41.98	51.69
CodeLLaMa-13B + EventRL (ARG-F1)	80.88	50.51	65.69	62.56	41.39	51.97
CodeLLaMa-13B + EventRL (AVG-F1)	76.98	48.18	62.58	60.86	42.37	51.62
CodeLLaMa-13B + EventRL (Prod-F1)	77.03	50.78	63.91	62.57	42.14	52.35
CodeLLaMa-34B + SFT	74.65	56.69	65.67	57.98	39.52	48.75

Table 1: Performance comparison of different Large Language Models (LLMs), including LLaMa, CodeLLaMa, and GPT4 on the ACE05 dataset. We highlight the best performance in dark gray and the second-best in light gray. We evaluate FSP method with GPT4 using 0, 1, 2, 3 example settings, where examples are randomly sampled from the training set. For FSP (0-Shot), both Held-in and Held-out tests involve unseen event types, while for other FSP methods with examples, Held-in tests use seen event types, and Held-out tests use unseen event types. Additionally, we compare the performance of CodeLLaMa-34B with SFT, noting that due to limited computational resources, the EventRL method was not applied to CodeLLaMa-34B.

loss. SFT experiments were primarily conducted on **LLaMa** variants (LLaMa2-7B and LLaMa2-13B) and **CodeLLaMa** models (7B, 13B, and 34B versions). **(3) EventRL with Proposed Reward Functions:** Our proposed EventRL framework was evaluated in three distinct configurations, each utilizing a different reward function designed to optimize the model's performance in event extraction: **EventRL (Arg-F1)** utilizes Argument-F1 as feedback, **EventRL (Avg-F1)** aims to balance Trigger-F1 and Argument-F1, and **EventRL (Prod-F1)** seeks to maximize the product of Trigger-F1 and Argument-F1. These variants were tested across **LLaMa** (Touvron et al., 2023) and **CodeLLaMa** (Roziere et al., 2023) models to investigate their effectiveness in event extraction.

Implementation details of different methods can be found in Appendix A.2.

5 Experimental Results

5.1 Main Results

Overall Performance Table 1 presents a comprehensive performance comparison of different LLMs, including GPT4, LLaMa, and CodeLLaMa variants, across various training approaches such as SFT (Supervised Fine-Tuning), FSP (Few-Shot Prompting), and our proposed EventRL methods. The table reports Trigger and Argument F1 scores, alongside their averages (denoted as AVG), which are calculated as the mean of the respective Trigger and Argument F1 scores for each method, both for events seen during training (Held-in test) and for novel, unseen event types (Held-out test). Our findings show that EventRL outperforms both SFT and FSP methods in event extraction. Specifically, when comparing the AVG scores, EventRL demonstrates superior overall performance. For example, in the Held-in test, the EventRL (Prod-F1) method using the LLaMa-7B model achieves an AVG score of 60.72, surpassing the SFT method's 56.03. Similarly, in the Held-out test, EventRL (Prod-F1) with LLaMa-7B reaches an AVG score of 40.84, compared to 37.35 by SFT. This indicates EventRL's effectiveness in accurately identifying event structures and predicting events.

Moreover, EventRL shows remarkable general-

ization capabilities, especially in handling unseen event types. Using the LLaMa-13B model, EventRL (Prod-F1) scores an AVG of 44.41 in the Held-out test, outperforming the SFT method's 42.04. These results highlight EventRL's robustness and its ability to adapt to new, unseen event types better than the other methods. The success of EventRL can be attributed to its specialized reward functions (Arg-F1, AVG-F1, and Prod-F1), which provide targeted feedback for refining the model's understanding and extraction of events. This tailored approach ensures that EventRL not only excels in extracting events from texts but also adapts effectively across different model sizes and architectures, including LLaMa and CodeLLaMa variants.

Impact of Different Reward Functions As shown in Table 1, the choice of reward functions significantly influences the performance of LLMs in event extraction, with AVG-F1 and Prod-F1 rewards demonstrating clear advantages. For the LLaMa-7B model, the Prod-F1 reward function yielded the best AVG Score, reaching 60.72 in Held-in test and 40.84 in Held-out test. This indicates that focusing on the interdependence of trigger and argument performance through the Prod-F1 reward enhances the model's overall ability to accurately extract events. In the case of the larger LLaMa-13B model, the AVG-F1 reward function achieved an AVG Score of 65.90 in Held-in test, while the Prod-F1 function excelled in Held-out test with an AVG Score of 44.41. This performance trend is also reflected in the CodeLLaMa models, where the Prod-F1 reward again demonstrated its effectiveness, particularly achieving a 52.35 AVG Score in the Held-out test for the CodeLLaMa-13B model. These results underline the importance of carefully selecting reward functions to optimize the event extraction capabilities of LLMs, with Prod-F1 and AVG-F1 rewards proving to be particularly beneficial in fostering a deeper understanding and extraction of events from texts.

5.2 Further Analysis

Ablation Study Table 2 presents an ablation study focusing on key features of the EventRL approach, particularly looking at the impact of removing Teacher-Forcing and Advantage-Clipping on event extraction performance using the ACE05 dataset. As can be seen, removing Teacher-Forcing led to a significant performance drop, with the Trigger-F1 score decreasing from 73.06 to 65.38 and the Argument F1 score from 42.34 to 27.25 for the Argument-F1. Similarly, excluding Advantage-Clipping resulted in a decline, notably in the Argument-F1 score, from 42.34 to 39.68. These results highlight the critical role of both strategies in ensuring EventRL's effectiveness and stability for event extraction. More analysis of these two components can be found in Appendix A.3.

Method	Held-in test		Held-out test	
	Trig.	Arg.	Trig.	Arg.
SFT	71.33	40.74	48.51	26.18
EventRL (Arg-F1)	73.06	42.34	51.15	29.32
w/o Tearcher-Force	65.38	27.25	47.34	19.28
w/o Advantage-Clip	67.60	39.68	46.37	24.21
EventRL (Prod-F1)	72.03	49.41	51.71	29.97
w/o Tearcher-Force	67.81	36.01	46.24	16.12
w/o Advantage-Clip	72.20	44.60	49.11	21.24

Table 2: Ablation Study on ACE05 dataset using LLaMa-7B.

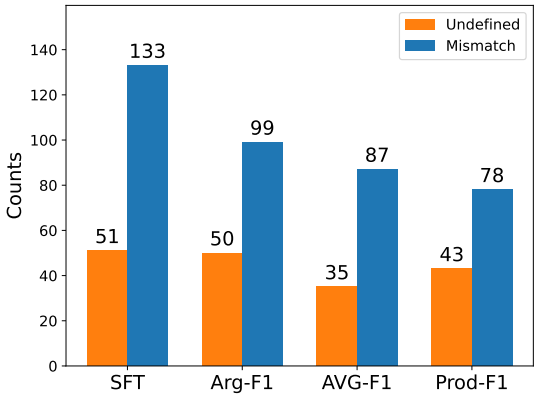


Figure 4: This chart quantifies the error counts for undefined event types and structural mismatches in event extraction on the LLaMa-7B model, comparing SFT with three EventRL training methods: Arg-F1, AVG-F1, and Prod-F1.

Error Analysis Figure 4 presents a comparative analysis of error types in event extraction when utilizing different training methods on the LLaMa-7B model. The Supervised Fine-Tuning (SFT) method resulted in a notably high occurrence of *undefined event type* errors, totaling 133, and structural mismatch errors, at 51. In contrast, the EventRL (Arg-F1) reduced "Undefined" errors to 99 and marginally decreased "Mismatch" errors to 50. A more significant improvement is observed with the EventRL (AVG-F1) approach, which cut down "Undefined" errors to 87 and "Mismatch" errors to the lowest count of 35, indicating a superior balance in error mitigation. The EventRL

(Prod-F1) also demonstrated improvement, lowering "Undefined" errors to 78 and "Mismatch" errors to 43, although not as effectively as AVG-F1. These numbers highlight the effectiveness of the EventRL training methods in reducing errors in large language model training for event extraction.

Code Data Pretraining Enhances Event Extraction Performance Table 1 reveals that models enhanced with code data, notably CodeLLaMa, significantly outperform their counterparts without code enhancement, like LLaMa, in event extraction. Specifically, at the 7B scale, CodeLLaMa (SFT) achieved an AVG score of 59.23 in Held-in test and 49.74 in Held-out test, surpassing LLaMa's 56.03 and 37.35, respectively. This improvement illustrates the positive impact of coding capabilities on the model's ability to extract events, both seen and unseen. When scaling up to 13B, the gap in performance between CodeLLaMa and LLaMa widens further, especially in the Held-out test, where CodeLLaMa (SFT) scores an AVG of 51.69, compared to LLaMa's 42.04. Additionally, employing the EventRL method with CodeLLaMa leads to superior outcomes across different reward function setups, demonstrating that code data enhancement not only boosts the model's understanding of structured information but also enhances its adaptability and accuracy in complex tasks like event extraction.

Analysis on Model Scale From Table 1, we observe a clear trend that increasing model scale positively impacts event extraction performance. For instance, when comparing the EventRL (Prod-F1) approach, the performance in terms of the AVG score improves significantly as we move from a 7B parameter model to a 13B parameter model, from 60.72 to 65.90 in the Held-in test. This indicates that larger models have enhanced capabilities in processing and understanding complex language structures, which leads to more accurate event extraction. However, scaling the model size further to 34B parameters introduces the risk of overfitting, especially evident in the Held-out test. For example, the CodeLLaMa-34B model, under the SFT approach, shows an AVG score of 65.67 in Held-in test but drops to 48.75 in Held-out test, indicating a decline in the model's ability to generalize to unseen event types.

Case Study Figure 5 showcases a comparison between the results of LLaMa-7B + SFT and LLaMa-7B + EventRL (Prod-F1). The input text describes the arrest of six individuals by British anti-terror police in Derbyshire and London. While both methods correctly identified the "ArrestJail" event, the location ("Derbyshire", "London"), and the action ("arrested"), LLaMa-7B + EventRL (Prod-F1) demonstrated a significant improvement by accurately including "police" as the agent conducting the arrest. Unlike LLaMa-7B + SFT, which missed the agent's role and the "crime" argument, LLaMa-7B + EventRL (Prod-F1)'s result reflects a comprehensive understanding of the event's structure, indicating its superior capability in capturing crucial aspects of events. More case studies can be found in Appendix A.3.

Figure 5: A comparison of event extraction results between LLaMa-7B + SFT and LLaMa-7B + EventRL (Prod-F1). Note that here the results of EventRL (Prod-F1) are totally accurate.

6 Conclusion

In this work, we demonstrated that EventRL, a reinforcement learning approach, significantly enhances the performance of LLMs in event extraction. By focusing on outcome supervision and utilizing specialized reward functions, EventRL effectively addresses the challenges of instruction following and hallucination in event extraction, leading to more accurate and reliable event extraction. The method's success is evident in its superior performance across various model sizes and architectures, particularly in handling novel event types. The importance of selecting appropriate reward functions and the positive impact of code data enhancement on event extraction capabilities have also been highlighted. Furthermore, our findings suggest that while increasing model scale can improve performance, there is a need to balance this with the ability to generalize to avoid overfitting.

Limitations

While EventRL is effective in event extraction for LLMs, it faces certain limitations. Firstly, the success of this method heavily depends on the availability of high-quality, well-balanced datasets and meticulous annotation efforts, which can be challenging and resource-intensive. Secondly, as the data volume increases, the training process becomes more time-consuming, necessitating advanced training frameworks and superior hardware capabilities to manage the computational demands efficiently. Lastly, EventRL is specifically designed to address the intricacies of event extraction tasks and does not inherently enhance the general capabilities of large models across a broader spectrum of NLP tasks. This focus on a niche area, while beneficial for its intended purpose, means that the improvements in event understanding and extraction may not translate to a broader enhancement of the models' overall performance in diverse linguistic tasks.

References

Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, et al. 2023. Gpt-4 technical report. *ArXiv preprint*, abs/2303.08774.

Samy Bengio, Oriol Vinyals, Navdeep Jaitly, and Noam Shazeer. 2015. Scheduled sampling for sequence prediction with recurrent neural networks. In *Advances in Neural Information Processing Systems 28: Annual Conference on Neural Information Processing Systems 2015, December 7-12, 2015, Montreal, Quebec, Canada*, pages 1171–1179.

Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, et al. 2021. Evaluating large language models trained on code. *ArXiv preprint*, abs/2107.03374.

Yubo Chen, Liheng Xu, Kang Liu, Daojian Zeng, and Jun Zhao. 2015. Event extraction via dynamic multi-pooling convolutional neural networks. In *Proc. of ACL*, pages 167–176, Beijing, China. Association for Computational Linguistics.

Yunmo Chen, Tongfei Chen, Seth Ebner, Aaron Steven White, and Benjamin Van Durme. 2020. Reading the manual: Event extraction as definition comprehension. In *Proceedings of the Fourth Workshop on Structured Prediction for NLP*, pages 74–83, Online. Association for Computational Linguistics.

Walker Christopher, Strassel Stephanie, Medero Julie, and Maeda Kazuaki. ACE 2005 Multilingual Training Corpus. https://catalog.ldc.upenn.edu/LDC2006T06.

Ganqu Cui, Lifan Yuan, Ning Ding, Guanming Yao, Wei Zhu, Yuan Ni, Guotong Xie, Zhiyuan Liu, and Maosong Sun. 2023. Ultrafeedback: Boosting language models with high-quality feedback. *ArXiv preprint*, abs/2310.01377.

Xinya Du and Claire Cardie. 2020. Event extraction by answering (almost) natural questions. In *Proc. of EMNLP*, pages 671–683, Online. Association for Computational Linguistics.

Jun Gao, Huan Zhao, Changlong Yu, and Ruifeng Xu. 2023a. Exploring the feasibility of chatgpt for event extraction. *ArXiv preprint*, abs/2303.03836.

Jun Gao, Huan Zhao, Yice Zhang, Wei Wang, Changlong Yu, and Ruifeng Xu. 2023b. Benchmarking large language models with augmented instructions for fine-grained information extraction. *ArXiv preprint*, abs/2310.05092.

Kuan-Hao Huang, I Hsu, Tanmay Parekh, Zhiyu Xie, Zixuan Zhang, Premkumar Natarajan, Kai-Wei Chang, Nanyun Peng, Heng Ji, et al. 2023a. A reevaluation of event extraction: Past, present, and future challenges. *ArXiv preprint*, abs/2311.09562.

Lei Huang, Weijiang Yu, Weitao Ma, Weihong Zhong, Zhangyin Feng, Haotian Wang, Qianglong Chen, Weihua Peng, Xiaocheng Feng, Bing Qin, et al. 2023b. A survey on hallucination in large language models: Principles, taxonomy, challenges, and open questions. *ArXiv preprint*, abs/2311.05232.

Leslie Pack Kaelbling, Michael L Littman, and Andrew W Moore. 1996. Reinforcement learning: A survey. *Journal of artificial intelligence research*, 4:237–285.

Timo Kaufmann, Paul Weng, Viktor Bengs, and Eyke Hüllermeier. 2023. A survey of reinforcement learning from human feedback. *ArXiv preprint*, abs/2312.14925.

Fayuan Li, Weihua Peng, Yuguang Chen, Quan Wang, Lu Pan, Yajuan Lyu, and Yong Zhu. 2020. Event extraction as multi-turn question answering. In *Findings of the Association for Computational Linguistics: EMNLP 2020*, pages 829–838, Online. Association for Computational Linguistics.

Junyi Li, Jie Chen, Ruiyang Ren, Xiaoxue Cheng, Wayne Xin Zhao, Jian-Yun Nie, and Ji-Rong Wen. 2024. The dawn after the dark: An empirical study on factuality hallucination in large language models. *ArXiv preprint*, abs/2401.03205.

Junyi Li, Xiaoxue Cheng, Wayne Xin Zhao, Jian-Yun Nie, and Ji-Rong Wen. 2023. Halueval: A large-scale hallucination evaluation benchmark for large language models. In *Proc. of EMNLP*, pages 6449–6464.

Hunter Lightman, Vineet Kosaraju, Yura Burda, Harri Edwards, Bowen Baker, Teddy Lee, Jan Leike, John Schulman, Ilya Sutskever, and Karl Cobbe. 2023. Let's verify step by step. *ArXiv preprint*, abs/2305.20050.

Xiao Liu, Zhunchen Luo, and Heyan Huang. 2018. Jointly multiple events extraction via attention-based graph information aggregation. In *Proc. of EMNLP*, pages 1247–1256, Brussels, Belgium. Association for Computational Linguistics.

Jie Lou, Yaojie Lu, Dai Dai, Wei Jia, Hongyu Lin, Xianpei Han, Le Sun, and Hua Wu. 2023. Universal information extraction as unified semantic matching. *ArXiv preprint*, abs/2301.03282.

Yaojie Lu, Hongyu Lin, Jin Xu, Xianpei Han, Jialong Tang, Annan Li, Le Sun, Meng Liao, and Shaoyi Chen. 2021. Text2Event: Controllable sequence-to-structure generation for end-to-end event extraction. In *Proc. of ACL*, pages 2795–2806, Online. Association for Computational Linguistics.

Yaojie Lu, Qing Liu, Dai Dai, Xinyan Xiao, Hongyu Lin, Xianpei Han, Le Sun, and Hua Wu. 2022. Unified structure generation for universal information extraction. In *Proc. of ACL*, pages 5755–5772, Dublin, Ireland. Association for Computational Linguistics.

Thien Huu Nguyen, Kyunghyun Cho, and Ralph Grishman. 2016. Joint event extraction via recurrent neural networks. In *Proc. of NAACL-HLT*, pages 300–309, San Diego, California. Association for Computational Linguistics.

Long Ouyang, Jeffrey Wu, Xu Jiang, Diogo Almeida, Carroll Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, et al. 2022. Training language models to follow instructions with human feedback. *Advances in Neural Information Processing Systems*, 35:27730–27744.

Baolin Peng, Chunyuan Li, Pengcheng He, Michel Galley, and Jianfeng Gao. 2023. Instruction tuning with gpt-4. *ArXiv preprint*, abs/2304.03277.

Steven J. Rennie, Etienne Marcheret, Youssef Mroueh, Jerret Ross, and Vaibhava Goel. 2017. Self-critical sequence training for image captioning. In *2017 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2017, Honolulu, HI, USA, July 21-26, 2017*, pages 1179–1195. IEEE Computer Society.

Baptiste Roziere, Jonas Gehring, Fabian Gloeckle, Sten Sootla, Itai Gat, Xiaoqing Ellen Tan, Yossi Adi, Jingyu Liu, Tal Remez, Jérémy Rapin, et al. 2023. Code llama: Open foundation models for code. *ArXiv preprint*, abs/2308.12950.

Oscar Sainz, Iker García-Ferrero, Rodrigo Agerri, Oier Lopez de Lacalle, German Rigau, and Eneko Agirre. 2023. Gollie: Annotation guidelines improve zero-shot information-extraction. *ArXiv preprint*, abs/2310.03668.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. 2017. Proximal policy optimization algorithms. *ArXiv preprint*, abs/1707.06347.

Nisan Stiennon, Long Ouyang, Jeffrey Wu, Daniel M. Ziegler, Ryan Lowe, Chelsea Voss, Alec Radford, Dario Amodei, and Paul F. Christiano. 2020. Learning to summarize with human feedback. In *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*.

Richard S Sutton and Andrew G Barto. 2018. *Reinforcement learning: An introduction*. MIT press.

Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, et al. 2023. Llama 2: Open foundation and fine-tuned chat models. *ArXiv preprint*, abs/2307.09288.

Jonathan Uesato, Nate Kushman, Ramana Kumar, Francis Song, Noah Siegel, Lisa Wang, Antonia Creswell, Geoffrey Irving, and Irina Higgins. 2022. Solving math word problems with process-and outcome-based feedback. *ArXiv preprint*, abs/2211.14275.

David Wadden, Ulme Wennberg, Yi Luan, and Hannaneh Hajishirzi. 2019. Entity, relation, and event extraction with contextualized span representations. In *Proc. of EMNLP*, pages 5784–5789, Hong Kong, China. Association for Computational Linguistics.

Xiao Wang, Weikang Zhou, Can Zu, Han Xia, Tianze Chen, Yuansen Zhang, Rui Zheng, Junjie Ye, Qi Zhang, Tao Gui, et al. 2023a. Instructuie: Multi-task instruction tuning for unified information extraction. *ArXiv preprint*, abs/2304.08085.

Yizhong Wang, Swaroop Mishra, Pegah Alipoormolabashi, Yeganeh Kordi, Amirreza Mirzaei, Atharva Naik, Arjun Ashok, Arut Selvan Dhanasekaran, Anjana Arunkumar, David Stap, Eshaan Pathak, Giannis Karamanolakis, Haizhi Lai, Ishan Purohit, Ishani Mondal, Jacob Anderson, Kirby Kuznia, Krima Doshi, Kuntal Kumar Pal, Maitreya Patel, Mehrad Moradshahi, Mihir Parmar, Mirali Purohit, Neeraj Varshney, Phani Rohitha Kaza, Pulkit Verma, Ravsehaj Singh Puri, Rushang Karia, Savan Doshi, Shailaja Keyur Sampat, Siddhartha Mishra, Sujan Reddy A, Sumanta Patro, Tanay Dixit, and Xudong Shen. 2022. Super-NaturalInstructions: Generalization via declarative instructions on 1600+ NLP tasks. In *Proc. of EMNLP*, pages 5085–5109, Abu Dhabi, United Arab Emirates. Association for Computational Linguistics.

Zhen Wang, Rameswar Panda, Leonid Karlinsky, Rogerio Feris, Huan Sun, and Yoon Kim. 2023b. Multitask prompt tuning enables parameter-efficient transfer learning. *ArXiv preprint*, abs/2303.02861.

Kaiwen Wei, Xian Sun, Zequn Zhang, Jingyuan Zhang, Guo Zhi, and Li Jin. 2021. Trigger is not sufficient: Exploiting frame-aware knowledge for implicit event argument extraction. In *Proc. of ACL*, pages 4672–4682, Online. Association for Computational Linguistics.

Xiang Wei, Xingyu Cui, Ning Cheng, Xiaobin Wang, Xin Zhang, Shen Huang, Pengjun Xie, Jinan Xu, Yufeng Chen, Meishan Zhang, et al. 2023. Zero-shot information extraction via chatting with chatgpt. *ArXiv preprint*, abs/2302.10205.

Weiwen Xu, Deng Cai, Zhisong Zhang, Wai Lam, and Shuming Shi. 2023. Reasons to reject? aligning language models with judgments. *ArXiv preprint*, abs/2312.14591.

Sen Yang, Dawei Feng, Linbo Qiao, Zhigang Kan, and Dongsheng Li. 2019. Exploring pre-trained language models for event extraction and generation. In *Proc. of ACL*, pages 5284–5294, Florence, Italy. Association for Computational Linguistics.

Fei Yu, Anningzhe Gao, and Benyou Wang. 2023. Outcome-supervised verifiers for planning in mathematical reasoning. *ArXiv preprint*, abs/2311.09724.

Zhiyuan Zeng, Jiatong Yu, Tianyu Gao, Yu Meng, Tanya Goyal, and Danqi Chen. 2023. Evaluating large language models at evaluating instruction following. *ArXiv preprint*, abs/2310.07641.

Shengyu Zhang, Linfeng Dong, Xiaoya Li, Sen Zhang, Xiaofei Sun, Shuhe Wang, Jiwei Li, Runyi Hu, Tianwei Zhang, Fei Wu, et al. 2023a. Instruction tuning for large language models: A survey. *ArXiv preprint*, abs/2308.10792.

Yue Zhang, Yafu Li, Leyang Cui, Deng Cai, Lemao Liu, Tingchen Fu, Xinting Huang, Enbo Zhao, Yu Zhang, Yulong Chen, et al. 2023b. Siren's song in the ai ocean: A survey on hallucination in large language models. corr abs/2309.01219 (2023).

Wayne Xin Zhao, Kun Zhou, Junyi Li, Tianyi Tang, Xiaolei Wang, Yupeng Hou, Yingqian Min, Beichen Zhang, Junjie Zhang, Zican Dong, et al. 2023. A survey of large language models. *ArXiv preprint*, abs/2303.18223.

Jeffrey Zhou, Tianjian Lu, Swaroop Mishra, Siddhartha Brahma, Sujoy Basu, Yi Luan, Denny Zhou, and Le Hou. 2023. Instruction-following evaluation for large language models. *ArXiv preprint*, abs/2311.07911.

Yang Zhou, Yubo Chen, Jun Zhao, Yin Wu, Jiexin Xu, and Jinlong Li. 2021. What the role is vs. what plays the role: Semi-supervised event argument extraction via dual question answering. In *Thirty-Fifth AAAI Conference on Artificial Intelligence, AAAI 2021, Thirty-Third Conference on Innovative Applications of Artificial Intelligence, IAAI 2021, The Eleventh Symposium on Educational Advances in Artificial Intelligence, EAAI 2021, Virtual Event, February 2-9, 2021*, pages 14638–14646. AAAI Press.

A Appendix

A.1 Dataset

Dataset	Count
Train	285
Dev	64
Held-in Test	125
Held-out Test	380

Table 3: Dataset sample counts based on our partitioning of the ACE05 dataset, with a focus on maintaining balance across event types. It is important to note that individual samples may contain multiple event types, leading to shared samples among different event types. This approach ensures a balanced representation of event types, despite variations in sample counts for each type.

Event Type	Count
Conflict.Attack	1244
Movement.Transport	608
Life.Die	515
Contact.Meet	254
Personnel.End-Position	170
Transaction.Transfer-Money	167
Personnel.Elect	153
Life.Injure	121
Contact.Phone-Write	112
Transaction.Transfer-Ownership	109
Personnel.Start-Position	107
Justice.Charge-Indict	101
Justice.Trial-Hearing	97
Justice.Sentence	93
Justice.Arrest-Jail	79

Table 4: Top 15 event types by sample count in the ACE05 dataset.

Dataset Details To provide a detailed and comprehensive evaluation of our model, we conducted a series of experiments using the ACE05 dataset, widely recognized for its variety in event types. This diversity enables us to rigorously test the model's ability to recognize and extract both familiar (seen) and novel (unseen) event types with precision. The ACE05 dataset comprises 33 distinct event types, serving as a robust foundation for our experimental framework.

Our experimental design involved the careful selection of event types for different portions of the dataset: the training set, validation set, and the test sets. Specifically, we chose 7 event types for inclusion in the training and validation sets, as

well as the held-in test set. These event types were selected based on their prevalence in the dataset, as indicated by the Table 4. This decision was informed by the observation that the dataset exhibits a significant imbalance in the distribution of event types, with the top 7 event types each having more than 150 samples, while the least represented event type has as few as 2 samples. To ensure a rigorous evaluation of the model's generalization capabilities, we selected 19 different event types to construct the held-out test set, focusing on unseen event types.

To address the challenge of data imbalance and to maintain a balanced dataset, we adopted a sampling strategy that ensures equitable representation of each event type across different sets. Specifically, we sampled 50 instances for each event type in the training set, 10 for each in the validation set, and 20 for each in both the held-in and held-out test sets. This balanced approach ensures that the model is exposed to and evaluated under a variety of conditions, offering a comprehensive insight into its performance across a wide range of event types. Table 3 shows the statistics of our ACE05 dataset split. We aimed for balance among event types, although some samples might include several events. This method ensures a fair representation across the dataset.

Event Extraction Guidelines Our work is built upon the foundational work of Sainz et al. (2023), which introduced a Python code-based representation for input and output in information extraction tasks. The essence of this work lies in the integration of event type extraction guidelines into the prompt, enhancing zero-shot generalization capabilities.

Our model operates on a schema defined in Python classes, with docstrings providing guidelines and comments outlining representative annotation candidates. This structured format ensures clarity, facilitates parsing, and aligns with modern Large Language Models' (LLMs) pretraining on code datasets. The output, beginning after "result =", comprises a list of class instances, yielding a transparent and easily parsable structure when executed in Python.

A complete example of our event definitions is displayed in Figure 12 within the Held-in Test, showcasing the guidelines used for extracting event types. We have adapted and refined the data processing code, originally available on the GitHub repository [1], to accommodate our specific needs for event extraction.

A.2 Implementation Details

Our implementation of EventRL comprises three pivotal components: Few-Shot Prompting (FSP) with GPT-4, Supervised Fine-Tuning (SFT), and the EventRL framework.

GPT-4 and Few-Shot Prompting (FSP) For the Few-Shot Prompting experiments, we utilized the GPT-4 API provided by Azure OpenAI Service [2], version dated 2023-05-15. Our experiments spanned four settings: 0-shot, 1-shot, 2-shot, and 3-shot. In the 0-shot setup, we did not include any demonstration examples in the instructions. For the 1-shot, 2-shot, and 3-shot setups, we introduced 1, 2, and 3 demonstration examples into the instructions, respectively. We tested across three distinct instruction templates and chose the one that showed the best overall performance for our results' presentation.

Supervised Fine-Tuning (SFT) The SFT experiments were conducted on various models: LLaMa-7B/13B and CodeLLaMa-7B/13B/34B. We used the ColossalAI [3] framework on two A100 servers. The training setup was as follows: learning rate set to 2e-5, with a minimum learning rate of 2e-6, weight decay at 0.1, micro batch size of 2, global batch size of 64, using bf16 for mixed precision, over 10 training epochs. For the 7B and 13B models, the parallel strategy involved a Tensor Parallel Size of 2 and a Pipeline Parallel Size of 2. The 34B model's strategy was adjusted to a Tensor Parallel Size of 4 and a Pipeline Parallel Size of 2. We selected the best model checkpoint for final use based on its average Trigger-F1 and Argument-F1 scores on the validation set.

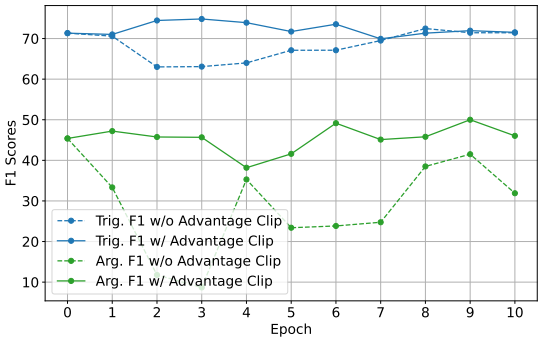
EventRL Leveraging the SFT groundwork, we proceeded to further train LLaMa-7B/13B and CodeLLaMa-7B/13B models using the EventRL method. The EventRL training was not applied to the 34B model due to computational limits. For the base model selection in EventRL training, we considered the 7B model's lower overfitting risk and the 13B model's higher risk, ultimately choosing the best SFT model for the 7B and the checkpoint

[1] https://github.com/hitz-zentroa/GoLLIE

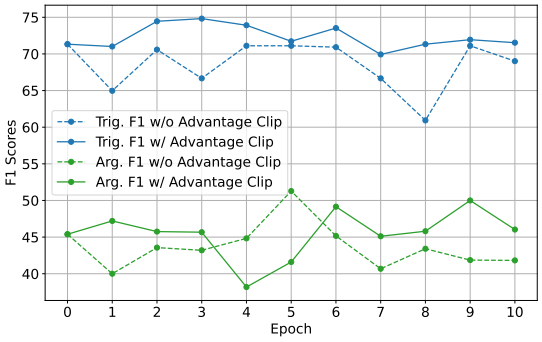
[2] https://learn.microsoft.com/en-us/azure/ai-services/openai/overview

[3] https://github.com/hpcaitech/ColossalAI/tree/main

from the previous epoch of the best SFT model for the 13B. The EventRL was implemented with the Huggingface transformers [4] framework, setting the training parameters as follows: a learning rate of 5e-6, a global batch size of 32, a micro batch size of 2, spanning 10 training epochs, using bf16 for mixed precision, and applying advantage clipping at 10. We set the teacher forcing threshold at 70 for the 7B model and 30 for the 13B model. The parameters for generating results with the sampling method were a temperature of 0.5 and a top p of 0.95.



(a)



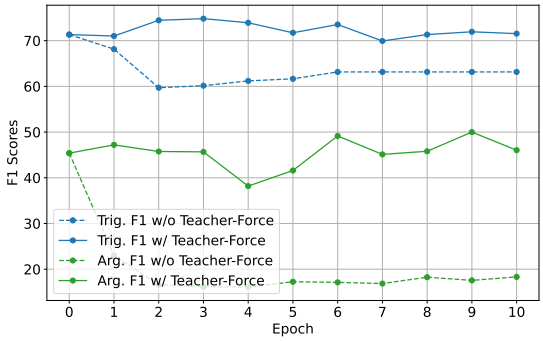
(b)

Figure 6: Training performance of LLama-7B model on EventRL with and without Advantage Clipping. Figure (a) shows the results for EventRL (Arg-F1) process, and Figure (b) for EventRL (Prod-F1) process. Both graphs compare the Trigger Extraction (Trig. F1) and Argument Extraction (Arg. F1) F1 scores over 10 epochs, illustrating the impact of Advantage Clipping on model stability and learning consistency.

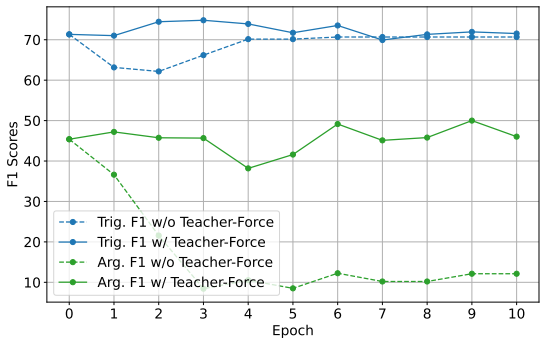
A.3 More Analysis on EventRL

Analysis on Teacher Force Threshold The Teacher-Force Threshold appears to be a pivotal strategy for stabilizing the training process in EventRL. By examining the Figure 7a and Figure 7b, we can infer that the models with Teacher-Force (denoted by "w/ Teacher-Force" in the legends) maintain or improve performance consistently across epochs, as opposed to models trained without Teacher-Force (denoted by "w/o Teacher-Force"), which exhibit more significant fluctuations and generally lower performance scores.

[4] https://github.com/huggingface/transformers



(a)



(b)

Figure 7: Training performance of the LLama-7B model employing EventRL strategies over epochs. Figure (a) shows the results using the EventRL (Arg-F1) method, while Figure (b) utilizes the EventRL (Prod-F1) method. Both figures track the Trigger F1 (Trig. F1) and Argument F1 (Arg. F1) metrics across ten epochs. The performance is measured on a validation set

In Figure 7a, which shows the EventRL (Arg-F1) model's performance, the presence of Teacher-Forcing corresponds with higher and more stable Argument F1 scores. The stability is particularly notable in the later epochs, suggesting that Teacher-Forcing aids the model in retaining knowledge over time, likely mitigating the effects of catastrophic forgetting. Similarly, in Figure 7b for the EventRL (Prod-F1) method, we see that the use of Teacher-Forcing correlates with more stable Trigger F1 scores. The Trigger F1 performance without Teacher-Force drops noticeably after the first few epochs, while with Teacher-Force, the performance remains relatively stable.

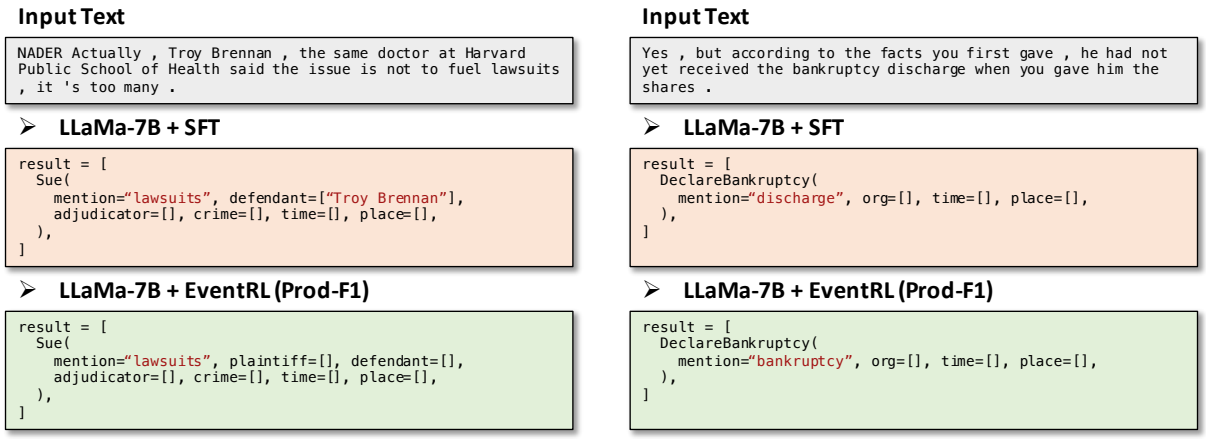


Figure 8: Comparative analysis of event extraction outcomes between LLaMa-7B + SFT and LLaMa-7B + EventRL (Prod-F1) on two distinct cases. Case 1 involves a discussion on lawsuits without explicitly assigning roles, where EventRL accurately avoids misattributing roles unlike SFT. Case 2 focuses on bankruptcy proceedings, with EventRL correctly identifying the primary event mention as "bankruptcy". Note that here the results of EventRL (Prod-F1) are totally accurate.

	Held-in test			Held-out test		
Method	**Trigger**	**Argument**	**AVG**	**Trigger**	**Argument**	**AVG**
GPT4+FSP (0-shot) + Prompt1	4.65	22.43	13.54	0.00	5.66	2.83
GPT4+FSP (0-shot) + Prompt2	6.04	22.08	14.06	15.42	17.69	16.56
GPT4+FSP (0-shot) + Prompt3	1.47	21.36	11.41	3.59	18.39	10.99
GPT4+FSP (1-shot) + Prompt2 + Example1	23.02	22.82	22.92	19.32	17.83	18.58
GPT4+FSP (1-shot) + Prompt2 + Example2	8.63	24.37	16.50	8.87	19.08	13.97
GPT4+FSP (1-shot) + Prompt2 + Example3	15.44	23.33	19.38	12.62	19.50	16.06

Table 5: Performance comparison of GPT-4 with Few-Shot Prompting (FSP) using different prompts and examples across "Held-in test" and "Held-out test" settings. The table showcases the Trigger and Argument F1 scores, along with the average (AVG) performance for each method.

Analysis on Advantage Clipping For both Argument Extraction (Arg. F1) and Trigger Extraction (Trig. F1), the results suggest that Advantage Clipping contributes to more consistent performance across epochs. Specifically, in Figure 6a, the use of Advantage Clipping appears to maintain higher F1 scores for Argument Extraction throughout the training epochs, with less variability compared to the setting without Advantage Clipping. The presence of Advantage Clipping seems to prevent drastic drops in performance, which could be indicative of the model retaining previously learned information better, thus mitigating catastrophic forgetting. Similarly, in Figure 6b, for Trigger Extraction, the application of Advantage Clipping demonstrates a more stable and consistently higher performance curve than without it. The variance is visibly reduced, which suggests that Advantage Clipping allows each sample to influence the learning process enough to be remembered, but not so much that it causes significant performance swings.

The most notable observation is the presence of fewer and less severe dips in performance for both Argument and Trigger F1 scores when Advantage Clipping is applied. This smoothing effect implies that Advantage Clipping indeed sets a floor for learning contributions from each sample, ensuring that all training data is utilized effectively, which is particularly crucial for a model that might otherwise focus too heavily on the most recent or the most rewarding examples.

Case Study As shown in Figure 8, we analyze two instances where the EventRL (Prod-F1) model outperforms the LLaMa-7B + SFT approach. In the first case, the input text discusses a statement made by Troy Brennan from Harvard Public School of Health regarding lawsuits. The LLaMa-7B + SFT model inaccurately identifies Troy Brennan as a defendant in a lawsuit, reflecting a misunderstanding of the context. In contrast, the LLaMa-7B +

EventRL (Prod-F1) model provides a more accurate representation by not assigning specific roles to the entities involved in the "Sue" event. This output aligns better with the input text, which does not explicitly define plaintiffs or defendants but rather discusses the broader issue of lawsuits.

The second input text refers to a situation involving bankruptcy proceedings. The LLaMa-7B + SFT model incorrectly identifies "discharge" as the event mention, which misrepresents the text's focus. The term "discharge" in the context of bankruptcy refers to the legal process of releasing a debtor from certain obligations, but the key event is the declaration of bankruptcy itself. The LLaMa-7B + EventRL (Prod-F1) model correctly identifies "bankruptcy" as the event mention, providing a more accurate and relevant extraction.

A.4 More Analysis on GPT4+FSP

Analyzing Various Prompts for GPT4+FSP (0-shot) The performance of GPT-4 with Few-Shot Prompting (0-shot) varies significantly across different prompt templates, illustrating the impact of instructional design on the model's ability to extract events. Prompt 2 (See Figure 10), which provides a clear and structured task description along with an explicit output format, yields the highest overall performance, especially noticeable in the "Held-out test" section with a notable average score of 16.56. This suggests that the clarity and specificity of the instructions can significantly enhance the model's understanding and execution of the task. In contrast, Prompts 1 (See Figure 9) and 3 (See Figure 11), despite offering detailed instructions, do not match the effectiveness of Prompt 2, potentially due to differences in how the task and output format are communicated.

Analyzing Different Examples for GPT4+FSP (1-shot) Introducing examples in the Few-Shot Prompting (1-shot) setup with GPT-4 shows a nuanced effect on performance, underscoring the influence of example selection. The inclusion of Example 1 with Prompt 2 significantly boosts performance across both "Held-in test" and "Held-out test", achieving an average score of 18.58 in the latter. This improvement indicates that the right example can enhance the model's understanding of the task, leading to better event extraction outcomes. However, the impact of different examples varies, with Example 2 and Example 3 leading to mixed results.

A.5 Discussion

One key observation from our study is the difference in performance between large and small models. Large models tend to perform better because they have more capacity to understand and process complex information. This means they can better identify the events in texts and figure out the relationships between different parts of the information. However, not everyone can use large models because they need a lot of computing power and resources. This is where our EventRL framework comes into play. EventRL is designed to help smaller models perform better on complex tasks. It does this by focusing on the outcomes of the task and using reinforcement learning to guide the model towards better performance.

With EventRL, even smaller models can improve their ability to extract events from texts. The approach helps these models pay closer attention to the final goal of the task and learn from each attempt. This way, they get better over time at understanding what's important in the text and how to accurately identify events and their details.

 # Event Definitions
{definitions}

 # Instruction
Your task is to analyze the given text and extract any relevant events based on the provided event data structures. After analyzing the text, you should output the results as a Python list of dataclass instances, each representing an identified event. Ensure that each instance is properly filled with the corresponding attributes according to the event type. If certain information is not available in the text for any attribute, leave the corresponding list empty.
The output should be in the following format:
```python
result = [
    Elect(mention="election", person=[], entity=[], position=[], time=[], place=[])
]
```

- Input:
{input}
- Output:

Figure 9: Template 1 illustrates a structured approach to event extraction, combining Python dataclass definitions with clear natural language instructions. It emphasizes a precise output format, guiding users on how to represent extracted events as a list of dataclass instances.

 # Event Definitions
{definitions}

 # Instruction
Your task is to analyze the given text and extract any relevant events based on the provided event data structures. After analyzing the text, you should output the results as a Python list of dataclass instances, each representing an identified event. Ensure that each instance is properly filled with the corresponding attributes according to the event type. If certain information is not available in the text for any attribute, leave the corresponding list empty.
The output should be in the following format:
```python
result = [
    EventDataClass(mention="", arg1=[], arg2=[], ...)
]
```
In this template, replace EventDataClass with the actual class name for the event, fill in the mention with the event trigger from the text, and populate arg1, arg2, etc., with the relevant information as lists, according to the details extracted from the text.

- Input:
{input}
- Output:

Figure 10: Template 2 presents a hybrid format that merges programming structure with user-friendly instructions for event extraction. It details how to fill out dataclass instances based on text analysis, with a focus on maintaining a clear and standardized output format.

 # Event Definitions
{definitions}

 # Instruction
Your task involves analyzing a provided text to identify and extract specific events, using predefined data structures for different event types. For each identified event, you are to create an instance of the appropriate data class, accurately populating its fields based on the information extracted from the text. The mention field, a string, should describe the key phrase or term that indicates the event, while other fields, which will vary depending on the event type, should be filled with lists of relevant details (e.g., names, places, times) as extracted from the text. If the text does not supply information for a particular field, you should leave that field's list empty.

Compile your findings into a Python list of these data class instances, following the format below:

```python
result = [
EventDataClass(mention="trigger phrase", arg1=["detail1", "detail2"],
arg2=["detail1"], ...)
]
```

In this template, replace EventDataClass with the specific class name that corresponds to the identified event. The mention should be filled with the text that clearly indicates the event, and arg1, arg2, etc., should be populated with the extracted details, formatted as lists, according to the data structure of the event type identified.

- Input:
{input}
- Output:

Figure 11: Template 3 offers a comprehensive guide for identifying and extracting events from text, using predefined data structures. It specifies how to populate data class fields with extracted details, aiming to enhance clarity and accuracy in representing events through a list of dataclass instances.

```python
@dataclass
class Attack(ConflictEvent):
    """An Attack Event is defined as a violent physical act causing harm or damage. Attack Events include any
    such Event not covered by the Injure or Die subtypes, including Events where there is no stated agent.
    """
    mention: str  # The text span that most clearly expresses (triggers) the event
    attacker: List[str]  # The attacking/instigating agent
    target: List[str]  # The target of the attack (including unintended targets)
    instrument: List[str]  # The instrument used in the attack
    time: List[str]  # When the attack takes place
    place: List[str]  # Where the attack takes place

@dataclass
class Transport(MovementEvent):
    """A Transport Event occurs whenever an Artifact (Weapon or Vehicle) or a Person is moved from one Place
    (GPE, Facility, Location) to another."""
    mention: str  # The text span that most clearly expresses (triggers) the event
    agent: List[str]  # The agent responsible for the transport Event
    artifact: List[str]  # The person doing the traveling or the artifact being traveled
    vehicle: List[str]  # The vehicle used to transport the person or artifact
    price: List[str]  # The price of transporting the person or artifact
    origin: List[str]  # Where the transporting originated
    destination: List[str]  # Where the transporting is directed
    time: List[str]  # When the transporting takes place

@dataclass
class Die(LifeEvent):
    """A Die Event occurs whenever the life of a Person Entity ends. Die Events can be accidental, intentional
    or self-inflicted"""
    mention: str  # The text span that most clearly expresses (triggers) the event
    agent: List[str]  # (Optional) The attacking agent / The killer
    victim: List[str]  # The person(s) who died
    instrument: List[str]  # The device used to kill
    time: List[str]  # When the death takes place
    place: List[str]  # Where the death takes place

@dataclass
class Meet(ContactEvent):
    """A Meet Event occurs whenever two or more Entities come together at a single location and interact with
    one another face-to-face. Meet Events include talks, summits, conferences, meetings, visits, and any
    other Event where two or more parties get together at some location."""
    mention: str  # The text span that most clearly expresses (triggers) the event
    entity: List[str]  # The agents who are meeting
    time: List[str]  # When the meeting takes place
    place: List[str]  # Where the meeting takes place

@dataclass
class TransferMoney(TransactionEvent):
    """TransferMoney Events refer to the giving, receiving, borrowing, or lending money when it is not in the
    context of purchasing something. The canonical examples are: (1) people giving money to organizations
    (and getting nothing tangible in return); and (2) organizations lending money to people or other orgs.
    """
    mention: str  # The text span that most clearly expresses (triggers) the event
    giver: List[str]  # The donating agent
    recipient: List[str]  # The recipient agent
    beneficiary: List[str]  # The agent that benefits from the transfer
    money: List[str]  # The amount given, donated or loaned
    time: List[str]  # When the amount is transferred
    place: List[str]  # Where the transation takes place

@dataclass
class EndPosition(PersonnelEvent):
    """An EndPosition Event occurs whenever a Person Entity stops working for (or changes offices within) an
    Organization or GPE."""
    mention: str  # The text span that most clearly expresses (triggers) the event
    person: List[str]  # The employee
    entity: List[str]  # The employer
    position: List[str]  # The JobTitle for the position being ended
    time: List[str]  # When the employment relationship ends
    place: List[str]  # Where the employment relationship ends

@dataclass
class Elect(PersonnelEvent):
    """An Elect Event occurs whenever a candidate wins an election designed to determine the Person argument
    of a StartPosition Event."""
    mention: str  # The text span that most clearly expresses (triggers) the event
    person: List[str]  # The person elected
    entity: List[str]  # The voting agent(s)
    position: List[str]  # The JobTitle for the position being nominated to
    time: List[str]  # When the election takes place
    place: List[str]  # Where the election takes place
```

Figure 12: ACE05 Event definitions in Held-in test.