BoundRL: Efficient Token-level Structured Text Segmentation through Reinforced Boundary Generation

Haoyuan Li[1]*, **Zhengyuan Shen**[2]†, **Sullam Jeoung**[2], **Yueyan Chen**[2]

Jiayu Li[2], **Qi Zhu**[2] , **Shuai Wang**[2], **Vassilis Ioannidis**[2], **Huzefa Rangwala**[2] ‡

[1]University of North Carolina at Chapel Hill, [2]Amazon Web Services

haoyuanl@cs.unc.edu; {donshen, sullamij, jlijiayu, qzhuamzn, wshui, ivasilei, rhuzefa}@amazon.com

Abstract

Structured texts refer to texts containing structured elements beyond plain texts, such as code snippets and placeholders. Such structured texts increasingly require segmentation into semantically meaningful components, which cannot be effectively handled by conventional sentence-level segmentation methods. To address this, we propose BoundRL, a novel approach that jointly performs efficient token-level text segmentation and label prediction for long structured texts. Instead of generating full texts for each segment, it generates only starting tokens and reconstructs the complete texts by locating these tokens within the original texts, thereby reducing output tokens by 90% and minimizing hallucination. To train the models for the boundary generation, BoundRL performs reinforcement learning with verifiable rewards (RLVR) that jointly optimizes document reconstruction fidelity and semantic alignment. It further mitigates entropy collapse by constructing intermediate candidates by perturbing segment boundaries and labels to create stepping stones toward higher-quality solutions. Experiments show that BoundRL enables small language models (1.7B parameters) to outperform few-shot prompting with much larger models as well as SFT and standard RLVR baselines on complex prompts used for LLM applications.

1 Introduction

Text segmentation is the task of dividing a text into coherent segments, each covering a distinct topic (Hearst, 1994). Beyond identifying segment boundaries, some approaches also predict the topic of each segment (Arnold et al., 2019a). These segments can help readers better understand the structure of long texts (Jeoung et al., 2026), QA systems retrieve more relevant contexts (Wang et al., 2025), prompt optimization system more efficiently optimize long prompts (Schnabel and Neville, 2024a), and summarization systems summarize long documents (Moro and Ragazzi, 2022).

Most previous works (Hearst, 1994; Lukasik et al., 2020) perform text segmentation on the sentence or paragraph level. These methods assume that texts can be cleanly divided into sentences or paragraphs. However, this assumption breaks down for many structured texts, such as LLM prompts, that include tables, code snippets, and placeholders. These elements generally do not conform to conventional sentence or paragraph structure. A natural solution is to adapt these methods to perform text segmentation at the token level. However, sequence labeling on the token level tends to generate very fragmented segments, while boundary classification requires an impractically large number of classifications for each token. Recently, Schnabel and Neville (2024b) performed token-level segmentation by instructing LLMs to generate the full text of each segment. However, they face high inference costs and hallucination risk for long texts due to regenerating the entire input (Wang et al., 2024).

We introduce BoundRL, a novel approach that jointly performs token-level text segmentation and label prediction specifically designed for long structured texts, which we term *structured text segmentation*. As shown in Fig. 1, BoundRL reformulates structured text segmentation as *boundary generation* and only generates a sequence of starting tokens and a label for each segment, then reconstructs full segments by locating them in the input. Unlike methods that generate every segment in full, this formulation reduces output tokens during inference by 90%, shifting complexity from linear in text length to linear in the number of segments. It also mitigates hallucination risks.

Boundary generation training presents unique optimization challenges. Supervised fine-tuning

*Work done during an internship at Amazon Web Service.
†Corresponding Author
‡work done at Amazon

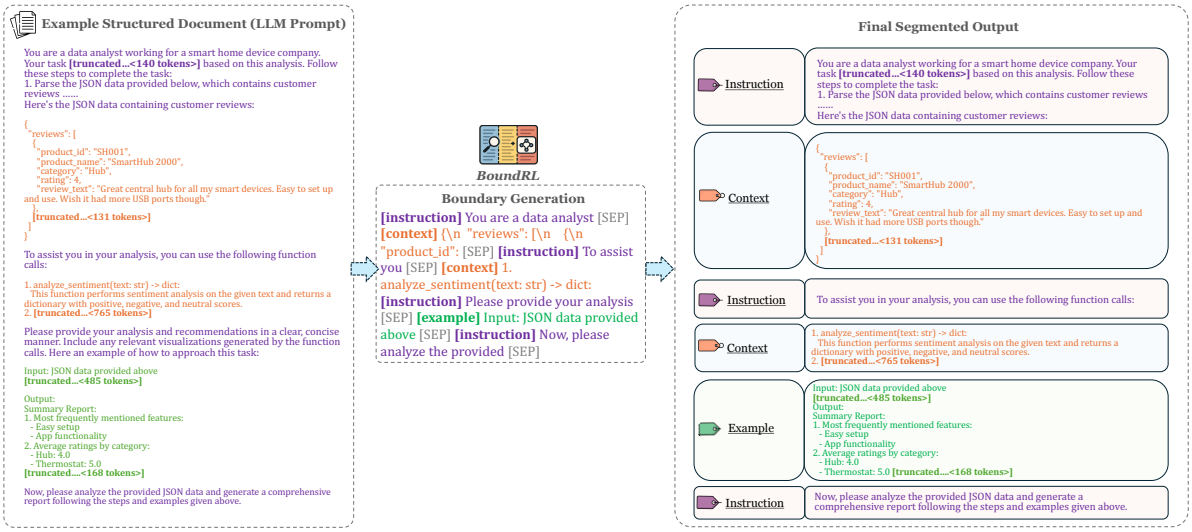


Figure 1: Efficient output pattern used by BoundRL. Instead of generating full segment text, it only generates starting tokens for each segment and then reconstructs full segments by locating these tokens in the original text, thereby reducing inference costs by orders of magnitude and minimizing hallucination. Note how the output length is independent of input document length.

(SFT) can mistakenly penalize starting tokens that correspond to the right boundary positions and provides insufficient penalties for minor token mismatches that cause failures in locating starting tokens. BoundRL addresses this by reinforcement learning with verifiable rewards (RLVR) (Shao et al., 2024), optimizing a reward function with two complementary dimensions: *reconstruction fidelity*, which measures whether the text can be fully recovered from generated segments, and *semantic alignment* which evaluates agreements between generated segments and annotated segments. This reward design ensures that different starting tokens corresponding to the same boundary receive identical rewards, while starting tokens with minor mismatches receive smaller rewards.

However, RLVR can suffer from entropy collapse (Cui et al., 2025), where generated sequences of segments become trapped in narrow, low-reward regions during rollout. Although annotated sequences of segments can provide high-reward examples, they are often too distant from the model's current generation distribution to enable effective learning. To mitigate this, BoundRL constructs intermediate candidates by perturbing generated sequences of segments through boundary adjustments and label modifications as shown in Fig. 2, creating stepping stones that bridge the gap between current generations and optimal solutions. This approach is particularly effective for our reward function due to its dense, continuous nature.

To evaluate BoundRL on particularly challenging structured texts, we construct StructSeg, a comprehensive dataset for structured text segmentation with 15.3K annotations of synthetic prompts and prompts from LangSmith[1] with text and label of each segment. Our evaluation focuses on prompts for LLMs due to their extreme structural complexity – dense mixtures of natural language instructions, code snippets, JSON formatting, and placeholders, making them unsuitable for sentence- or paragraph-level segmentation.

Our experiments show that relatively small models (1.7B-4B parameters) trained with BoundRL outperform few-shot prompting using much larger models (Claude-4 Sonnet (Anthropic, 2025)). Moreover, BoundRL brings significant performance and generalization improvement over SFT and the intermediate candidates can further improve the performance of RLVR.

Our contributions are four-fold:

- BoundRL, a boundary-generation approach for token-level structured text segmentation that reduces output tokens by 90% while mitigating hallucination risks;
- A dual-objective reward function optimizing reconstruction fidelity and semantic alignment;
- A selective perturbation strategy that addresses entropy collapse by generating learnable intermediate candidates during RLVR;
- StructSeg, a 15.3K human-annotated benchmark where BoundRL enables 1.7B models to

[1]https://smith.langchain.com/hub/

outperform much larger models such as Claude-4 Sonnet.

2 Related Work

Text segmentation Text segmentation aims to divide a text into coherent segments, where each segment encompasses a distinct semantic unit or topic (Hearst, 1994). The task has been studied across various domains, such as regular documents (Koshorek et al., 2018b) and dialogues (Xing and Carenini, 2021). Arnold et al. (2019a); Barrow et al. (2020) extend the task by jointly modeling segmentation and topic classification. In supervised settings, segmentation is often framed as sequence labeling, where each sentence is labeled as a boundary or not (Koshorek et al., 2018a; Li et al., 2018). Other works frame text segmentation as a boundary classification task predicting whether a sentence is a boundary based on its surrounding context (Lukasik et al., 2020). More recently, Inan et al. (2022); Duarte et al. (2024) frame the text segmentation as a generation task by generating the starting sentence or paragraph indices of each segment. However, these methods face limitations on structured texts containing elements like code snippets and JSON data formats, which lack traditional sentence or paragraph boundaries. Schnabel and Neville (2024b); Jeoung et al. (2026); Zhang et al. (2023) propose to segment texts using LLMs by generating the full text of each segment. However, they face high inference costs and hallucination risk for long texts. Given these challenges, BoundRL frame token-level structured text segmentation as boundary generation.

Reinforcement Learning with Verifiable Reward Compared with RLHF, which relies on a separate reward model to assign rewards (Ouyang et al., 2022), RLVR (Shao et al., 2024) uses a rule-based reward function. The design makes RLVR particularly efficient for structured text segmentation, as rewards can be easily computed by comparing generated and annotated segments. However, candidates generated during rollout suffer from being trapped in narrow, low-reward regions, known as entropy collapse (Cui et al., 2025). To mitigate this, Zhang et al. (2025) propose to generate candidates conditioned on the prefix of reference data with varying lengths. However, this requires generating as many candidates as there are segments in the input text, leading to high training costs for structured text segmentation. Yan et al. (2025); Dong et al. (2025) include reference candidates generated by a reference model, which can be too distant from the model's current generation distribution for effective learning. In contrast, BoundRL proposes to generate intermediate candidates which are closer to the current distribution of generation.

3 Problem Statement

Structured texts refer to texts containing structured elements beyond plain texts, such as code snippets. Structured text segmentation takes a structured text as input and generates a list of segments:

$$\begin{aligned} &[(l_1, t_1), ..., (l_n, t_n)] = f(d), \\ &s.t.\ l_i \neq l_{i-1}, t_i \cap t_{i-1} = \emptyset, \forall i = 2, ..., n \end{aligned} \tag{1}$$

where f denotes the system, d denotes the input structured text, t_i denotes the text of the i-th segment and $l_i \in L$ denotes the semantic label of the i-th segment. L denotes the set of potential labels for segments, which varies by domains. For simplicity, we use T^L to denote $[(l_i, t_i)_{i=1}^n]$. For our case study on prompts, labels include 'instruction', 'example', 'context', 'question', and 'output format' as shown in Tab. 5.

4 Method

The training process of BoundRL consists of two stages: SFT followed by RLVR. Sec. 4.1 describes adaptation of LLMs to an efficient output pattern via SFT. Sec.4.2 describes the reward design of RLVR. Sec. 4.3 describes the construction of intermediate candidates for the rollout stage of RLVR.

4.1 Efficient Output Pattern for Structured Text Segmentation

To enable efficient structured text segmentation, BoundRL formulates the task as boundary generation. Instead of regenerating full segment text, it produces only a sequence of starting tokens and a label for each segment, then reconstructs the segments from the input as shown in Fig. 1. Specifically, given an input structured text d, BoundRL tunes the LLM to generate a sequence of starting tokens s_i and a label l_i for each segment: $[(\hat{l}_i, \hat{s}_i)_{i=1:n}]$. To adapt LLMs for boundary generation, BoundRL transforms the annotated text of each segment t_i into corresponding starting token sequences s_i. The length of each starting token sequence is randomly sampled between 2 and 10 tokens. To reduce the chance that different segments share the same starting tokens, BoundRL increases

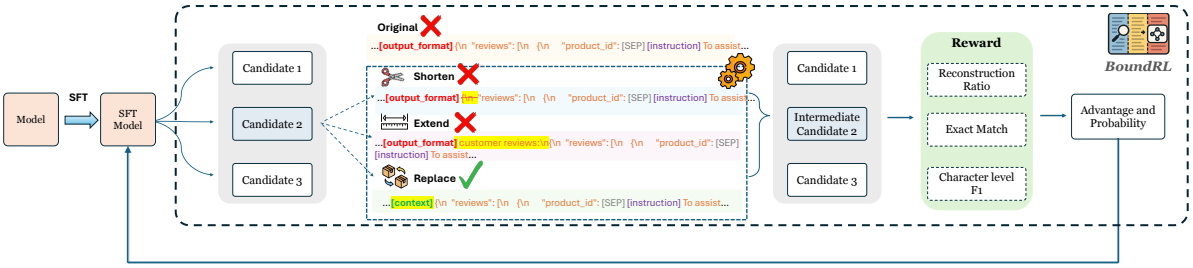


Figure 2: RLVR workflow of BoundRL showing the dual-objective reward function and intermediate candidate construction. The reward function combines reconstruction ratio for reconstruction fidelity with exact match and character-level F1 scores for semantic alignment. To mitigate entropy collapse during rollout, BoundRL constructs intermediate candidates by perturbing generated segments through boundary adjustments and label modifications.

the length of corresponding sequences of starting tokens until they become different or until the full segment length is reached. The LLMs are then fine-tuned using SFT on the starting tokens sequences and their corresponding labels.

BoundRL then reconstructs the text of each segment $\hat{t}_i$ using the position of each sequence of starting tokens $\hat{s}_i$ in the input structured text d. To prevent potential ambiguity among segments that share the same starting tokens, the reconstruction process operates iteratively. Specifically, for the first sequence of starting tokens $\hat{s}_1$, BoundRL locates it as its leftmost occurrence in the input d. For each subsequent sequence $\hat{s}_i$, BoundRL locates it as its leftmost occurrence in the input d **after** the position of its previous segment. This introduces an ordering constraint, where each match must occur later in the input text than the previous one. Therefore, even if the same starting token sequence appears multiple times, each occurrence is uniquely assigned to a segment based on its position in this left-to-right traversal. The text of the i-th segment $\hat{t}_i$ is then extracted as the text span between sequences of starting tokens $\hat{s}_i$ and $\hat{s}_{i+1}$. For simplicity, we use $\hat{T}^L$ to denote the sequence of reconstructed segment texts and their labels $[(\hat{l}_i, \hat{t}_i)_{i=1}^n]$. If the positions of either $\hat{s}_i$ or $\hat{s}_{i+1}$ cannot be found, the i-th segment will be discarded. Compared with regenerating full text of each segment, BoundRL reduces hallucination risks inherent in text generation while reducing required output tokens from $O(|d|)$ to $O(n)$ tokens, where $|d| \gg n$ denotes the input length.

4.2 Reward Design of RLVR

Boundary generation presents optimization challenges for SFT, as it can mistakenly penalize correct starting tokens and provides a insufficient penalty for minor token mismatches. Therefore, BoundRL applies RLVR after SFT using the reward function with two dimensions: *reconstruction fidelity*, which measures whether the input can be fully recovered from generated segments, and *semantic alignment* which evaluates agreements between generated and annotated segments.

To measure reconstruction fidelity, BoundRL uses the reconstruction ratio ρ_{rec}. The metric is calculated as the proportion of the input text d that can be successfully reconstructed from the texts of generated segments $\hat{t}_i$: $\rho_{\text{rec}}(\hat{T}^L, d) = \sum_{i=1}^n |\hat{t}_i|/|d|$, where $|*|$ denotes the length of a text in characters. The reconstruction ratio $\rho_{\text{rec}}(*)$ ranges from zero to one, with higher values indicating more complete reconstruction of the input structured text d. Compared with SFT, the metric imposes stronger penalties on segments with token mismatches.

To measure semantic alignment, BoundRL uses two metrics. The first metric is the F-1 score of exact match $\text{EM}(\hat{T}^L, T^L)$ between generated segments $\hat{T}^L$ and annotated segments T^L (Tjong Kim Sang and De Meulder, 2003). A generated segment $(\hat{l}_i, \hat{t}_i)$ is considered an exact match to an annotated segment (l_j, t_j) if their texts and labels are the same. The F1 score is then computed as the harmonic mean of precision (the fraction of generated segments an exact match) and recall (the fraction of annotated segments with an exact match). However, the F-1 score of exact match $\text{EM}(*)$ can be too strict since minor token-level differences between segment texts can lead to a mismatch. Therefore, BoundRL additionally uses the character-level F-1 score $\text{F1}_{\text{char}}(\hat{T}^L, T^L)$ motivated by part-of-speech (POS) tagging (Marcus et al., 1993). The metric treats structured text segmentation as a character-level labeling task, where each character in the input d is assigned a label from the set L, based on the segment it belongs to. Specifically, all characters in $\hat{t}_i$ are assigned

the label $\hat{l}_i$, and likewise for annotated segments. The metric is then calculated as the weighted F-1 score between character-level labels from generated segments and those from annotated segments. Both the F-1 score of exact match $\text{EM}(*)$ and the character-level F-1 score $\text{F1}_{\text{char}}(*)$ range from zero to one, with higher values indicating better alignment between generated and annotated segments. Compared with SFT, different starting tokens that correspond to the same boundary have the same value for both metrics.

The final reward $r(*)$ for the generated segments $\hat{T}^L$ is calculated using both dimensions:

$$r(\hat{T}^L) = \rho_{\text{rec}}(\hat{T}^L)\frac{(\text{EM}(\hat{T}^L) + \text{F1}_{\text{char}}(\hat{T}^L))}{2} \quad (2)$$

For simplicity, the equation omits the input text d and the annotated segments T^L. The reward encourages generated segments to accurately reproduce starting tokens for complete reconstruction while getting close to annotated segments for high-quality segmentation.

4.3 Construction of Intermediate Candidate

In this section, we describe how BoundRL constructs and incorporates intermediate candidates during the rollout stage of RLVR to mitigate entropy collapse. Specifically, BoundRL constructs intermediate candidates by perturbing generated candidate segmentations and selectively replaces the originally generated candidate segmentations with the intermediate candidates for the training.

An effective intermediate candidate should lie between generated and annotated segments to provide meaningful yet learnable guidance. To construct such intermediate candidates, BoundRL first generates m candidate segmentations for an input d using standard RLVR practice, denoted as $[\hat{T}_j^L]_{j=1}^m$. These candidate segmentations are ordered by descending reward $r(\hat{T}_j^L)$. BoundRL then perturbs the candidate segmentation with the medium-level reward: $\hat{T}_{\frac{m}{2}}^L$. As shown in Fig. 2, three types of perturbations are considered for each segment: (1) shortening the text $\hat{t}_i$ by truncating one word from either side, (2) extending the text $\hat{t}_i$ by including additional one word from either side, or (3) replacing the label $\hat{l}_i$ with an alternative from the set of potential labels L, excluding labels already assigned to neighboring segments. To shorten or extend the text of a segment $\hat{t}_i$, BoundRL modifies the starting token sequences $\hat{s}_i$ or $\hat{s}_{i+1}$ accordingly. Applying a single perturbation to each segment creates a pool of potential intermediate candidates, each differing from the original by exactly one perturbation. The potential intermediate candidate with the highest reward $r(*)$ is selected as the final intermediate candidate, denoted as $\tilde{T}_{\frac{m}{2}}^L$.

	Prompts	Tokens	Segments
Synthetic	15,132	900	6.1
Langchain	197	914	7.6

Table 1: Statistics of synthetic and real-world prompts.

To avoid performance degradation from off-policy intermediate candidates, BoundRL employs a selective replacement strategy. In each training batch, BoundRL replaces the original candidate segmentations with the medium-level reward with the intermediate candidate for at most k inputs. Replacement is allowed only when the reward of the intermediate candidate $r(\tilde{T}_{\frac{m}{2}}^L)$ is higher than the original reward $r(\hat{T}_{\frac{m}{2}}^L)$. If more than k inputs satisfy this criterion, BoundRL selects the top-k with the largest gains to preserve training stability while incorporating the most beneficial refinements. For the selected inputs, BoundRL uses the intermediate candidate with the remaining m-1 generated candidate segmentations in the following training; otherwise, all m generated candidates are used.

5 Experimental Setup

5.1 StructSeg

In this section, we describe StructSeg, which serve as a case study of structured text segmentation. Table 1 summarizes the dataset statistics. It contains synthetic and real-world prompts. The synthetic prompts are generated using Claude 3.5 Sonnet (Anthropic, 2024), balancing diversity and complexity. To ensure diversity, we implement a multi-faceted sampling strategy that draws from varied prompt types (e.g. system prompt), prompt modes (e.g. prompt template), and task types (e.g. classification) and other factors. Additionally, we encourage the generated prompts to include structural elements, such as nested JSON, which make the prompts unsuitable for sentence-level segmentation. Some prompts exceed 2,000 words. More details of synthetic prompts are in App. A.5. The real-world prompts are collected from Langchain-hub [2] as task templates with each prompt corresponding to a distinct task to ensure diversity.

After collecting synthetic and real-world

[2]https://smith.langchain.com/hub

prompts, we recruit a group of highly experienced human annotators to perform annotation. Each segment is assigned one of five labels: 'instruction', 'example', 'context', 'question', or 'output format' (Table 5) following Mao et al. (2025); Jeoung et al. (2026). To ensure high annotation quality, we develop step-by-step labeling instructions for annotators. Annotators should first decompose each prompt into mutually exclusive, non-overlapping segments. Placeholders are extracted separately if they are knowledge input, user questions, or contextual information. Then, annotators determine if each segment is an instruction or few-shot examples. If neither applied, annotators write a description of the segment and then select the most appropriate label from the remaining labels.

We denote the subset of synthetic prompts as `Synthetic` and the subset of real-world prompts as `Langchain`. Tab. 1 reports the statistics of these subsets, and Fig. 8 shows the distribution of segment labels. For the `Synthetic` subset, we use 14,732 prompts for training, 200 for validation, and 200 for testing. For the `Langchain` subset, all prompts are used exclusively for testing.

5.2 Implementation Details

We evaluate `BoundRL` on three LLMs: Qwen3-1.7b, Qwen3-4b (Yang et al., 2025), and Llama-3.1-8b-Instruct (Dubey et al., 2024). The training process has two stages:

Stage 1: SFT All LLMs are first fine-tuned on the training set of `StructSeg` for one epoch with a batch size of 16. We use learning rates of 2e-6 for Qwen3 models and 5e-7 for Llama-3.1-8b-Instruct.

Stage 2: RLVR SFT-tuned models are then tuned with RLVR using GRPO (Shao et al., 2024) without standard deviation-based reward scaling (Liu et al., 2025). To control computational costs, we use a randomly sampled 25% subset of the training data. Each training batch contains 6 input documents, with $m = 4$ candidate segmentations generated per input text using a temperature of 1.2 during rollout. For intermediate candidate construction, we apply selective replacement with model-specific thresholds: $k = 2$ for Qwen3-1.7b and Qwen3-4b, and $k = 1$ for Llama-3.1-8b-Instruct. Learning rates are 1e-6 for Qwen3 models and 2e-7 for Llama-3.1. We save checkpoints every 0.2 epochs and select the best model based on validation performance.

During inference, the temperature is set to 0. We tune the hyperparameters based on their performance on the validation set. More implementation details are in App. A.2.

6 Experimental Results

6.1 Evaluation of `BoundRL`

In this section, we perform a comprehensive evaluation of `BoundRL`. We consider the following training schemes: (i) SFT, where models are fine-tuned with SFT for one epoch to adapt the output pattern of `BoundRL`; (ii) SFT w/2 epochs, where models are fine-tuned with SFT for two epochs; (iii) NER, where models are fine-tuned for two epochs to predict the label for each token like named entity recognition (NER) (Li et al., 2020); (iv) SFT+RLVR, a two-stage fine-tuning procedure as in `BoundRL`, but without intermediate candidates; (v) SFT+RLVR$_{\text{w/ high temp.}}$, the same as SFT+RLVR but with a higher sampling temperature of 1.5 during rollout; (vi) RL-PLUS (Dong et al., 2025), which uses one sequence of annotated segments and three candidate segmentations during rollout. For the NER baseline, we also include a NER baseline fine-tuned on ModernBERT-large (Warner et al., 2024), which is the SOTA pretrained model using bidirectional encoder. Additionally, to show the necessity of token-level segmentation instead of sentence-level segmentation for structured text segmentation, we include a sentence-level oracle baseline (Oracle$_{\text{sent}}$), where each sentence is mapped to the most overlapping ground-truth label. The Oracle$_{\text{sent}}$ baseline serves as an upper bound for previous sentence-level segmentation methods (Inan et al., 2022; Duarte et al., 2024). Furthermore, we consider a few-shot prompting baseline, where the LLM is instructed to segment an input prompt according to the target taxonomy and a provided example (Schnabel and Neville, 2024a; Zhao et al., 2025; Jeoung et al., 2026). For this baseline, we use Claude3.5v2-sonnet (Anthropic, 2024) and Claude4-sonnet (Anthropic, 2025) following Jeoung et al. (2026). We consider two output patterns: (i) full, which outputs the complete text of each segment; (ii) start, which outputs only the starting tokens, like `BoundRL`. Implementation details and qualitative examples of these baselines are in App. A.3 and A.7 respectively.

For evaluation, we use the reconstruction ratio $\rho_{\text{rec}}(*)$, the F-1 score of exact match EM$(*)$ and the character-level F-1 score F1$_{\text{char}}(*)$ as described in Sec. 4.2. We additionally use character-level P_k score (Beeferman et al., 1999) which measures the quality of segment boundaries, with the win-

Method	Synthetic					Langchain					Avg
	ρ_{rec}	EM	P_k	F1_{lab}	F1_{char}	ρ_{rec}	EM	P_k	F1_{lab}	F1_{char}	
$\text{Oracle}_{\text{sent}}$	100.0	2.7	16.5	99.7	90.3	100.0	5.2	18.9	99.5	92.2	75.4
ModernBERT+NER	99.7	57.3	5.7	85.8	95.2	98.8	19.4	15.6	66.4	86.5	78.8
Prompting Baselines											
$\text{Claude3.5-Sonnet}_{\text{full}}$	78.3	14.3	15.4	79.8	70.2	55.8	11.0	21.4	62.4	47.5	58.2
$\text{Claude3.5-Sonnet}_{\text{start}}$	50.0	16.9	25.2	73.8	47.2	48.1	11.4	23.4	61.3	41.1	50.1
$\text{Claude4-Sonnet}_{\text{full}}$	97.2	22.1	11.3	82.2	88.2	80.3	13.8	18.1	65.5	68.3	68.8
$\text{Claude4-Sonnet}_{\text{start}}$	90.1	22.8	13.6	79.8	81.8	87.0	18.3	18.1	67.9	71.6	68.8
Qwen3-1.7b											
SFT	99.3	72.3	4.6	94.1	93.7	87.7	34.7	14.7	77.0	71.1	81.1
SFT w/2epochs	99.5	73.5	3.9	94.4	94.6	85.0	41.0	14.6	77.6	70.9	81.8
NER	100.0	34.1	6.5	81.5	94.8	100.0	8.9	19.9	57.4	86.7	73.7
SFT+RLVR	100.0	77.4	4.1	94.7	94.6	88.6	47.2	13.2	79.1	74.6	83.9
$\text{SFT+RLVR}_{\text{w/temp.}}$	100.0	77.2	4.0	94.6	95.0	90.4	44.6	14.4	79.1	75.4	83.8
RL-PLUS	99.8	73.9	4.5	94.4	94.3	91.5	42.9	13.4	79.5	76.5	83.5
BoundRL	99.9	77.3	4.1	94.8	94.8	90.6	47.3	12.2	79.8	76.8	**84.5**
Qwen3-4b											
SFT	99.7	71.6	5.3	94.9	92.8	93.1	41.6	12.3	80.2	78.8	83.5
SFT w/2epochs	99.7	73.0	4.3	95.2	94.2	91.3	40.7	12.1	83.6	78.2	84.0
NER	100.0	41.9	6.9	82.8	95.6	100.0	8.9	24.7	59.3	85.7	74.3
SFT+RLVR	99.7	77.6	4.6	94.6	93.7	92.7	52.4	10.6	82.3	82.1	86.0
$\text{SFT+RLVR}_{\text{w/temp.}}$	99.7	77.3	4.9	94.4	93.3	87.6	47.0	12.5	77.6	74.2	83.4
RL-PLUS	99.7	76.6	4.2	94.3	94.1	94.8	51.0	10.8	81.5	83.1	86.0
BoundRL	99.7	78.3	4.0	94.8	94.7	94.1	52.4	10.3	82.5	83.3	**86.6**
Llama-3.1-8b-Instruct											
SFT	99.6	71.8	4.9	94.3	93.4	95.9	28.4	13.4	79.7	80.7	82.5
SFT w/2epochs	99.9	72.8	4.5	94.3	94.2	95.6	31.9	13.2	78.9	79.5	82.9
NER	100.0	25.9	12.3	69.9	92.4	100.0	7.0	24.9	55.7	83.4	69.7
SFT+RLVR	100.0	73.9	4.1	94.7	94.6	96.4	40.2	11.7	77.3	82.1	84.3
$\text{SFT+RLVR}_{\text{w/temp.}}$	99.7	72.7	4.1	94.0	94.6	91.8	43.3	13.0	77.5	78.9	83.5
RL-PLUS	100.0	73.0	4.4	94.4	94.3	95.9	37.9	11.7	78.0	82.7	84.0
BoundRL	100.0	76.1	4.4	94.4	94.1	96.3	42.8	11.5	78.0	82.1	**84.8**

Table 2: Evaluation of BoundRL across LLMs and datasets. The best-performing method for each LLM is highlighted in **bold**. BoundRL consistently outperform both finetuning baselines and few-shot prompting with much larger LLMs. The improvements are particularly big on the Langchain subset, showing BoundRL's superior generalization to real-world, out-of-domain prompts.

dow width set to half the average length of annotated segments following standard practice. We also use F1_{lab}, which is the micro-F1 score that compares the predicted label of each generated segment with the label of the most overlapping annotated segment following Arnold et al. (2019b). Higher values of all metrics except P_k indicate better performance, while lower P_k values are better. Results are reported in percentage on the test set of the Synthetic subset and the Langchain subset, along with the average of one minus P_k and other metrics across both subsets in Table 2. We also evaluate BoundRL on Wikisection-city dataset (Arnold et al., 2019b) in App. A.9.

We observe that BoundRL consistently outperforms all baselines. The difference between BoundRL and the second best-performing method (SFT+RLVR) is statistically significant using paired t-test ($p < 0.05$), showing the importance of intermediate candidates for effective RLVR training. We show in App. A.6 that BoundRL can mitigate the entropy collapse issue of RLVR in App. A.6. In contrast, RL-PLUS, which uses annotated segments during rollout, has inconsistent results and can even hurt performance. This may be because annotated segments are too out-of-distribution to provide useful learning signals. Additionally, increasing temperature ($\text{SFT+RLVR}_{\text{w/temp.}}$) cannot further improve the performance, showing that the improvement brought by intermediate candidates is not merely from increased exploration space but guided exploration.

Furthermore, models fine-tuned with RLVR consistently outperform SFT-only models and the improvement becomes bigger on the Langchain subset. The difference in average scores between SFT+RLVR and SFT w/2epochs is statistically significant using paired t-test ($p < 0.05$). The results show that SFT is insufficient for optimal performance as it can mistakenly penalize starting tokens that correspond to the right boundaries and give insufficient penalties for minor token mismatches. To further investigate this, we analyze how only generated starting tokens that correspond to the ground-truth positions but differ lexically from the ground-truth tokens for the SFT w/2 epoch baseline. The proportion is 21% for Llama3.1-8b, 28% for Qwen3-1.7b, and 19% for Qwen3-4b. This result indicates that a substantial fraction of predictions identify the correct boundary positions but

Method	Synthetic					Langchain					Avg
	ρ_{rec}	EM	P_k	F1_{lab}	F1_{char}	ρ_{rec}	EM	P_k	F1_{lab}	F1_{char}	
Qwen3-1.7b											
BoundRL	99.9	77.3	4.1	94.8	94.8	90.6	47.3	12.2	79.8	76.8	**84.5**
BoundRL w/ 2steps	99.9	78.0	4.4	94.8	94.7	88.6	45.7	13.6	80.5	75.0	83.9
BoundRL w/o select	99.7	76.9	4.5	94.6	94.1	89.9	45.3	12.2	79.8	76.5	84.0
BoundRL w/o middle	99.6	77.7	4.5	94.9	94.2	89.7	44.5	12.9	78.8	75.5	83.7
Qwen3-4b											
BoundRL	99.7	78.3	4.0	94.8	94.7	94.1	52.4	10.3	82.5	83.3	**86.6**
BoundRL w/ 2steps	99.7	78.3	3.7	94.5	94.7	94.7	50.0	10.9	81.5	84.8	86.4
BoundRL w/o select	99.7	77.4	4.5	94.8	93.6	93.7	52.1	10.5	83.0	84.0	86.3
BoundRL w/o middle	99.7	77.7	4.2	94.8	94.2	94.6	50.7	11.0	81.4	84.1	86.2
Llama-3.1-8b-Instruct											
BoundRL	100.0	76.1	4.4	94.4	94.1	96.3	42.8	11.5	78.0	82.1	**84.8**
BoundRL w/2steps	100.0	76.6	4.3	94.4	94.6	94.4	38.4	12.2	77.9	81.5	84.1
BoundRL w/o select	99.9	75.7	4.3	94.3	94.6	94.7	41.8	11.6	78.7	81.5	84.5
BoundRL w/o middle	99.9	74.7	4.4	94.0	94.2	95.5	43.2	12.1	77.6	81.8	84.4

Table 3: Ablation study of BoundRL. The best-performing method of each LLM is in **bold**.

are penalized due to different tokens.

We note that the smallest model (Qwen3-1.7b) fine-tuned with BoundRL significantly outperforms the best-performing few-shot prompting baseline (Claude4-sonnet-full) with much more parameters. Besides, prompting baselines that generate full segment text require an average of 1,170 tokens per input prompt on the Synthetic subset, while BoundRL requires only 119 tokens, which corresponds to a 90% reduction in output tokens and a significant efficiency gain.

Although models fine-tuned with NER achieve high scores on F1_{char}, they generally achieve lower scores on EM, P_k and F1_{lab}. Analysis of the outputs shows that models fine-tuned with NER tend to generate fragmented and short segments, showing the effectiveness of framing structured text segmentation as a boundary generation task.

Despite having access to gold annotations, $\text{Oracle}_{\text{sent}}$ performs poorly on both EM and P_k. This is mainly because LLM prompts contain many code snippets and structured data formats (e.g., JSON) that do not conform to standard sentence boundaries. This demonstrates the necessity of token-level segmentation for structured text segmentation. Moreover, since $\text{Oracle}_{\text{sent}}$ serves as an upper bound for prior sentence-level segmentation methods, its weak performance further shows that these methods are fundamentally inadequate for handling structured text.

6.2 Ablation Study of BoundRL

In this section, we perform an ablation study of BoundRL. We consider the following ablated versions of BoundRL: (i) BoundRL w/ 2steps, which performs two perturbation steps when generating intermediate candidates; (ii) BoundRL w/o select, which uses an intermediate candidate for all input texts in a batch without selective replacement; (iii) BoundRL w/o middle, which generates intermediate candidates by perturbing a randomly sampled candidate segmentation instead of the one with the medium-level reward. More details of these ablated versions are in App. A.8. The results are in Tab. 3.

Tab. 3 shows that BoundRL outperforms all ablated versions. Specifically, applying multiple perturbations when generating intermediate candidates (BoundRL w/ 2steps) and incorporating them for all input texts (BoundRL w/o select) both hurt performance. The results show the importance of controlling the distance between the current generation and intermediate candidates, which aligns with our findings in Sec. 6.1 that directly using annotated segments does not improve performance.

	start	end	start+end
Qwen3-1.7b	**81.1**	75.6	74.8
Qwen3-4b	**83.5**	82.3	80.3
Llama-3.1-8b-Instruct	**82.5**	80.9	77.6

Table 4: Evaluation of different output patterns. The best-performing output pattern is highlighted in **bold**. SFT w/start, the output pattern used by BoundRL, consistently outperforms other patterns.

6.3 Evaluation of Output Patterns

In this section, we evaluate output patterns for structured text segmentation by comparing LLMs fine-tuned with SFT for different output patterns. We consider three output patterns: (i) start, which is used by BoundRL and outputs starting tokens of each segment; (ii) end, which outputs ending tokens of each segment; (iii) start+end, which outputs both starting and ending tokens of each segment. We show examples of these output patterns in App. A.4. We show average scores on Synthetic and Langchain subsets in Tab. 4 and full results in Tab. 6.

Tab. 4 shows that `start`, used by `BoundRL`, consistently outperforms other output patterns across LLMs and datasets, with particular advantages in exact match scores. In contrast, `start+end` performs worse than both `start` and `end`, although it is supposed to be more robust to token mismatches as it generates both boundaries of each segment. This suggests that requiring generation of both starting and ending tokens imposes an excessive learning burden that degrades performance.

7 Conclusions

We propose `BoundRL`, a novel framework that reformulates structured text segmentation as a boundary generation problem. `BoundRL` generates only the starting tokens of each segment, which substantially reduces inference costs and mitigates hallucination risks. To train models for boundary generation, `BoundRL` employs RLVR to jointly optimize reconstruction fidelity and semantic alignment, while our intermediate candidate construction strategy alleviates entropy collapse during training. In a challenging case study on LLM prompts, `BoundRL` consistently outperforms SFT and RLVR baselines, and enables 1.7B models to outperform Claude-4 Sonnet few-shot prompting while using 90% fewer output tokens. The boundary generation paradigm may extend to other structured text domains such as legal documents and technical specifications.

8 Limitations

While our approach demonstrates promising performance on out-of-distribution structured texts, it still relies on domain-specific annotated datasets and task-specific fine-tuning. This requirement limits its applicability in true zero-shot settings for entirely new document types where annotated data is unavailable. Future work may explore more data-efficient training strategies, such as improved annotation reuse or weak and semi-supervised learning, to further reduce reliance on human-labeled data.

9 Ethics Statement

The human annotations were collected through hired annotators from a data annotation service. Annotators were instructed to strictly refrain from including any biased, hateful, or offensive content towards any race, gender, sex, or religion. The annotations underwent audits by a separate group of annotators, achieving an 89% inter-annotator agreement. We use LLM to polish the writing of the paper.

References

Anthropic. 2024. Claude 3.5 sonnet.

Anthropic. 2025. Introducing claude 4.

Sebastian Arnold, Rudolf Schneider, Philippe Cudré-Mauroux, Felix A Gers, and Alexander Löser. 2019a. Sector: A neural model for coherent topic segmentation and classification. *Transactions of the Association for Computational Linguistics*, 7:169–184.

Sebastian Arnold, Rudolf Schneider, Philippe Cudré-Mauroux, Felix A. Gers, and Alexander Löser. 2019b. SECTOR: A neural model for coherent topic segmentation and classification. *Transactions of the Association for Computational Linguistics*, 7:169–184.

Joe Barrow, Rajiv Jain, Vlad Morariu, Varun Manjunatha, Douglas W Oard, and Philip Resnik. 2020. A joint model for document segmentation and segment labeling. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 313–322.

Doug Beeferman, Adam Berger, and John Lafferty. 1999. Statistical models for text segmentation. *Machine learning*, 34(1):177–210.

Steven Bird, Ewan Klein, and Edward Loper. 2009. *Natural language processing with Python: analyzing text with the natural language toolkit*. " O'Reilly Media, Inc.".

Ganqu Cui, Yuchen Zhang, Jiacheng Chen, Lifan Yuan, Zhi Wang, Yuxin Zuo, Haozhan Li, Yuchen Fan, Huayu Chen, Weize Chen, and 1 others. 2025. The entropy mechanism of reinforcement learning for reasoning language models. *arXiv preprint arXiv:2505.22617*.

Yihong Dong, Xue Jiang, Yongding Tao, Huanyu Liu, Kechi Zhang, Lili Mou, Rongyu Cao, Yingwei Ma, Jue Chen, Binhua Li, and 1 others. 2025. Rl-plus: Countering capability boundary collapse of llms in reinforcement learning with hybrid-policy optimization. *arXiv preprint arXiv:2508.00222*.

André Duarte, João Marques, Miguel Graça, Miguel Freire, Lei Li, and Arlindo Oliveira. 2024. Lumberchunker: Long-form narrative document segmentation. In *Findings of the Association for Computational Linguistics: EMNLP 2024*, pages 6473–6486.

Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Amy Yang, Angela Fan, and 1 others. 2024. The llama 3 herd of models. *arXiv e-prints*, pages arXiv–2407.

Marti A. Hearst. 1994. Multi-paragraph segmentation expository text. In *32nd Annual Meeting of the Association for Computational Linguistics*, pages 9–16, Las Cruces, New Mexico, USA. Association for Computational Linguistics.

Hakan Inan, Rashi Rungta, and Yashar Mehdad. 2022. Structured summarization: Unified text segmentation and segment labeling as a generation task. *arXiv preprint arXiv:2209.13759*.

Sullam Jeoung, Yueyan Chen, Yi Zhang, Shuai Wang, Haibo Ding, and Lin Lee Cheong. 2026. PromptPrism: A linguistically-inspired taxonomy for prompts. In *Findings of the Association for Computational Linguistics: EACL 2026*, pages 1168–1192, Rabat, Morocco. Association for Computational Linguistics.

Omri Koshorek, Adir Cohen, and Noam Mor Michael Rotman Jonathan Berant. 2018a. Text segmentation as a supervised learning task. In *Proceedings of NAACL-HLT*, pages 469–473.

Omri Koshorek, Adir Cohen, Noam Mor, Michael Rotman, and Jonathan Berant. 2018b. Text segmentation as a supervised learning task. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 2 (Short Papers)*, pages 469–473, New Orleans, Louisiana. Association for Computational Linguistics.

Jing Li, Aixin Sun, Jianglei Han, and Chenliang Li. 2020. A survey on deep learning for named entity recognition. *IEEE transactions on knowledge and data engineering*, 34(1):50–70.

Jing Li, Aixin Sun, and Shafiq Joty. 2018. Segbot: a generic neural text segmentation model with pointer network. In *Proceedings of the 27th International Joint Conference on Artificial Intelligence*, pages 4166–4172.

Zichen Liu, Changyu Chen, Wenjun Li, Penghui Qi, Tianyu Pang, Chao Du, Wee Sun Lee, and Min Lin. 2025. Understanding r1-zero-like training: A critical perspective. *arXiv preprint arXiv:2503.20783*.

Michal Lukasik, Boris Dadachev, Kishore Papineni, and Gonçalo Simões. 2020. Text segmentation by cross segment attention. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 4707–4716, Online. Association for Computational Linguistics.

Yuetian Mao, Junjie He, and Chunyang Chen. 2025. From prompts to templates: A systematic prompt template analysis for real-world llmapps. *arXiv preprint arXiv:2504.02052*.

Mitchell P. Marcus, Beatrice Santorini, and Mary Ann Marcinkiewicz. 1993. Building a large annotated corpus of English: The Penn Treebank. *Computational Linguistics*, 19(2):313–330.

Gianluca Moro and Luca Ragazzi. 2022. Semantic self-segmentation for abstractive summarization of long documents in low-resource regimes. In *Proceedings of the AAAI conference on artificial intelligence*, volume 36, pages 11085–11093.

Long Ouyang, Jeffrey Wu, Xu Jiang, Diogo Almeida, Carroll Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, and 1 others. 2022. Training language models to follow instructions with human feedback. *Advances in neural information processing systems*, 35:27730–27744.

Tobias Schnabel and Jennifer Neville. 2024a. Symbolic prompt program search: A structure-aware approach to efficient compile-time prompt optimization. In *Findings of the Association for Computational Linguistics: EMNLP 2024*, pages 670–686, Miami, Florida, USA. Association for Computational Linguistics.

Tobias Schnabel and Jennifer Neville. 2024b. Symbolic prompt program search: A structure-aware approach to efficient compile-time prompt optimization. *arXiv preprint arXiv:2404.02319*.

Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, Haowei Zhang, Mingchuan Zhang, YK Li, Yang Wu, and 1 others. 2024. Deepseekmath: Pushing the limits of mathematical reasoning in open language models. *arXiv preprint arXiv:2402.03300*.

Erik F. Tjong Kim Sang and Fien De Meulder. 2003. Introduction to the CoNLL-2003 shared task: Language-independent named entity recognition. In *Proceedings of the Seventh Conference on Natural Language Learning at HLT-NAACL 2003*, pages 142–147.

Leandro von Werra, Younes Belkada, Lewis Tunstall, Edward Beeching, Tristan Thrush, Nathan Lambert, Shengyi Huang, Kashif Rasul, and Quentin Gallouédec. 2020. TRL: Transformers Reinforcement Learning.

Noah Wang, Feiyu Duan, Yibo Zhang, Wangchunshu Zhou, Ke Xu, Wenhao Huang, and Jie Fu. 2024. PositionID: LLMs can control lengths, copy and paste with explicit positional awareness. In *Findings of the Association for Computational Linguistics: EMNLP 2024*, pages 16877–16915, Miami, Florida, USA. Association for Computational Linguistics.

Zhitong Wang, Cheng Gao, Chaojun Xiao, Yufei Huang, Shuzheng Si, Kangyang Luo, Yuzhuo Bai, Wenhao Li, Tangjian Duan, Chuancheng Lv, and 1 others. 2025. Document segmentation matters for retrieval-augmented generation. In *Findings of the Association for Computational Linguistics: ACL 2025*, pages 8063–8075.

Benjamin Warner, Antoine Chaffin, Benjamin Clavié, Orion Weller, Oskar Hallström, Said Taghadouini, Alexis Gallagher, Raja Biswas, Faisal Ladhak, Tom Aarsen, Nathan Cooper, Griffin Adams, Jeremy

Howard, and Iacopo Poli. 2024. Smarter, better, faster, longer: A modern bidirectional encoder for fast, memory efficient, and long context finetuning and inference. *Preprint*, arXiv:2412.13663.

Linzi Xing and Giuseppe Carenini. 2021. Improving unsupervised dialogue topic segmentation with utterance-pair coherence scoring. In *Proceedings of the 22nd Annual Meeting of the Special Interest Group on Discourse and Dialogue*, pages 167–177, Singapore and Online. Association for Computational Linguistics.

Jianhao Yan, Yafu Li, Zican Hu, Zhi Wang, Ganqu Cui, Xiaoye Qu, Yu Cheng, and Yue Zhang. 2025. Learning to reason under off-policy guidance. *arXiv preprint arXiv:2504.14945*.

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, and 1 others. 2025. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*.

Kaiyi Zhang, Ang Lv, Jinpeng Li, Yongbo Wang, Feng Wang, Haoyuan Hu, and Rui Yan. 2025. Stephint: Multi-level stepwise hints enhance reinforcement learning to reason. *arXiv preprint arXiv:2507.02841*.

Wenzheng Zhang, Sam Wiseman, and Karl Stratos. 2023. Seq2seq is all you need for coreference resolution. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 11493–11504, Singapore. Association for Computational Linguistics.

ChenZhuo Zhao, Ziqian Liu, Xinda Wang, Junting Lu, and Chaoyi Ruan. 2025. PMPO: Probabilistic metric prompt optimization for small and large language models. In *Findings of the Association for Computational Linguistics: EMNLP 2025*, pages 14728–14761, Suzhou, China. Association for Computational Linguistics.

A Appendix

A.1 Taxonomy Used for Prompts

In this section, we describe the taxonomy use by `StructSeg`for structured text segmentation. The label of each segment can be 'instruction', 'example', 'context', 'question', or 'output format' as shown in Tab. 5

A.2 Implementation Details of BoundRL

In this section, we describe additional implementation details of BoundRL. To help model better adapt to the prompt segmentation task, we give models a meta instruction in addition to the prompt to be segmented. The meta instruction includes a brief definition of each segment type and an example of required output format. The meta instruction that requires the model to output starting tokens of each segment is shown in Fig. 5.

For both SFT and reinforcement learning, BoundRL sets the maximum gradient norm as 0.1 and the weight decay as 0.01. BoundRL uses the linear learning rate scheduler with a warmup ratio of 0.03. We tune the hyperparameters of BoundRL in stages. We first select the hyperparameters of SFT based on the performance of models that are fine-tuned SFT on the validation set. With the SFT hyperparameters fixed, we then tune the hyperparameters of RLVR, followed by those for intermediate candidate construction, using the same process. The SFT and reinforcement learning are implemented using 'SFTTrainer' and 'GRPOTrainer' from the 'trl' package (von Werra et al., 2020).

During the rollout stage of reinforcement learning, we notice that candidate segmentation might contain repetitive end of response tokens or segments after end of response tokens, which might hurt the training stability if are directly used for training. To address the issue, all generated candidate segmentations are truncated at the first end of response token.

To shorten or extend the text of a segment $\hat{t}_i$, BoundRL modifies the starting token sequences $\hat{s}_i$ or $\hat{s}_{i+1}$ accordingly. Specifically, to shorten the text of a segment $\hat{t}_i$ on the left side by one word, BoundRL truncates the first word of the starting token sequence $\hat{s}_i$. To shorten the text of a segment $\hat{t}_i$ on the right side by one word, BoundRL prepends to $\hat{s}i+1$ the word immediately before it. To extend the text of a segment $\hat{t}_i$ on the right side by one word, BoundRL truncates the first word of the starting token sequence $\hat{s}_{i+1}$. To extend the text of a segment $\hat{t}_i$ on the left side by one word, BoundRL prepends to $\hat{s}i$ the word immediately before it. Therefore, When constructing intermediate candidates, BoundRLdoes not shorten or extending the text of a segment with only one word. We will also modify the neighboring starting token sequences accordingly if there is a overlap between starting token sequences after modifications.

A.3 Implementation Details of Baselines

In this section, we describe the implementation details of all baselines. For the prompting baseline using Claude3.5-sonnet-v2 and Claude4-sonnet, the prompts used by the models first give a detailed definition for each label used by our taxonomy and then instructs the model to extract segments following the definitions. As described in Sec. 6.1, the

Label	Definition
Instruction	Guidance on how to process and respond to queries.
Example	Examples of what the input and corresponding output should look like.
Context	Background information and context that the model needs to refer to.
Question	Queries or questions provided specifically by users
Output Format	The type, format, or style of the output

Table 5: Example taxonomy for structured prompt segmentation used in StructSeg. Different domains may employ alternative taxonomies appropriate to their document types and analysis needs.

models are required to output either the full text of each segment or the starting tokens of each segment as BoundRL. To help model better understand the required output format, the prompts also include a randomly sampled prompt and its expected output format from the training set of StructSeg. The prompt for outputting the full text of each segment is shown in Figs 4. The prompt for outputting the starting tokens of each segment is shown in Fig. 4. The temperature during inference is set to 0.

SFT and SFT w/2epochs use the same hyper-parameters as the SFT training stage of BoundRL. The label schema of our NER baseline is motivated by (Tjong Kim Sang and De Meulder, 2003), which uses 'B-X', 'I-X', and 'O'. However, since all tokens in a prompt belong to a segment in the text segmentation task, we use 'B-X' to represent the beginning token of each segment and 'I-X' to represent the remaining tokens of each segment. To predict the label of each token, the output of the final layer for each token is feed into a single-layer MLP following the common practice. Other hyper-parameters of the NER baseline is the same as the SFT training stage of BoundRL. For ModernBERT-Large+NER, it is tuned for two epochs with a learning rate of 1e-5. For RL-PLUS, we replace one originally generated candidate segmentation with a sequence of annotated segments. The temperature to control the weight of advantage function is 1.0. Unless otherwise specified, RL-PLUS, SFT+RLVR, and SFT+RLVR$_{\text{w/ high temp.}}$ all use GRPO without reward scaling based on standard deviation and the same hyper-parameters as BoundRL for a fair comparison.

For Oracle$_{\text{sent}}$, we first split the input text d into sentences using 'sent_tokenize' function from the nltk package (Bird et al., 2009). We then label each sentence as the ground-truth label with the maximum number of overlapping tokens. The final segment is obtained by merging neighboring sentences with the same label.

A.4 Implementation Details of Other Output Patterns

In this section, we provide implementation details for output patterns other than the output pattern used by BoundRL ('start'). Specifically, for the 'end' output pattern, the model should first generate a label and then a sequence of ending tokens for each segment. The text of the i-th segment $\hat{t}_i$ is then extracted as the text span between the positions of the $i-1$-th and i-th sequence of ending tokens. For the 'start+end', the model should first output a label and then output a sequence of starting tokens and a sequence of ending tokens. The text of the i-th segment $\hat{t}_i$ is then extracted as the text span between the positions of the i-th sequence of starting tokens and the i-th sequence of ending tokens. We show the meta instruction that requires the model to output ending tokens of each segment in Fig. 6 and meta instruction that requires the model to output both starting and ending tokens of each segment in Fig. 7. We also show an example text and corresponding expected outputs for different output patterns in Fig. 9. Furthermore, we also show the full results for different output patterns in Tab. 6.

A.5 Generation of Synthetic Prompts

When generating the synthetic prompts, we implement a multi-faceted sampling strategy that draws from varied prompt types (system prompt, user prompt, combined), prompt modes (prompt template, full prompt, hybrid), and task types (e.g. classification, summarization), placeholder formats (e.g., `{context}` or `{{context}}`), the number of examples (zero, one, few), prompt lengths, writing styles, and levels of details. The full list of the task types, writing styles, prompt lengths, format types, and level of details are shown in Tab. 7.

A.6 Standard Deviation of Rewards during Training

In this section, we compare the standard deviation of rewards of SFT+RLVR and BoundRL during

Method	Synthetic					Langchain					Avg
	ρ_{rec}	EM	P_k	F1_{lab}	F1_{char}	ρ_{rec}	EM	P_k	F1_{lab}	F1_{char}	
Qwen3-1.7b											
SFT w/start	99.3	72.3	4.6	94.1	93.7	87.7	34.7	14.7	77.0	71.1	**81.1**
SFT w/end	96.0	64.8	6.0	92.0	89.3	77.7	26.5	19.5	71.1	63.9	75.6
SFT w/start+end	96.6	59.5	7.9	90.8	86.7	84.5	20.2	17.9	69.9	65.2	74.8
Qwen3-4b											
SFT w/start	99.7	71.6	5.3	94.9	92.8	93.1	41.6	12.3	80.2	78.8	**83.5**
SFT w/end	98.1	67.8	4.6	93.7	92.5	91.5	39.9	13.0	79.0	78.5	82.3
SFT w/start+end	98.8	70.4	5.9	93.9	92.7	85.2	33.7	14.1	76.5	71.8	80.3
Llama-3.1-8b-Instruct											
SFT w/start	99.6	71.8	4.9	94.3	93.4	95.9	28.4	13.4	79.7	80.7	**82.5**
SFT w/end	98.7	65.9	5.6	93.8	92.5	93.5	31.5	13.6	75.6	76.6	80.9
SFT w/start+end	97.8	62.9	6.2	90.8	91.4	87.7	25.1	17.1	71.4	72.3	77.6

Table 6: Evaluation of different output patterns. The best-performing output pattern is highlighted in **bold**. SFT w/start, the output pattern used by BoundRL, consistently outperforms other patterns.

training to evaluate whether BoundRL can mitigate the entropy collapse issue of RLVR. Specifically, we show the curve of standard deviation of rewards among candidate segmentations generated during rollout along the training. We show the curve of SFT+RLVR and BoundRL in Fig.10. From the figure, we find that the standard deviation of rewards of SFT+RLVR quickly becomes very small, while that of BoundRL remains stable throughout training. The results show that intermediate candidates help BoundRL mitigate the entropy collapse issue of RLVR.

A.7 Qualitative Examples

In this section, we show qualitative examples of BoundRL and other baselines. Specifically, we show an example prompt, its corresponding annotated segments, the raw output and reconstructed segments for each method in Fig. 12.

To analyze the performance of $\text{Oracle}_{\text{sent}}$, we show sentences split by its sentence tokenizer from a json snippet in Fig. 11. As shown in the figure, the first half of json snippet and its previous sentence are mistakenly merged into the first sentence. Similarly, the second half of the json snippet and the list of functions are also mistakenly merged into the second sentence. The example shows that the sentence tokenizer cannot effectively handle structured elements that do not follow conventional sentence boundaries. These failures are confirmed quantitatively by our $\text{Oracle}_{\text{sent}}$ in Tab. 2. Despite having access to ground-truth labels, it achieves only 2.7 and 5.2 EM on Synthetic and Langchain respectively, with high P_k scores (16.5 and 18.9).

A.8 Implementation Details of Ablation Studies

In this section, we provide more implementation details of the ablated versions of BoundRL. In BoundRL w/ 2steps, we construct the intermediate candidates by selecting the first perturbation step that has the biggest reward gain over the original candidate segmentation with the medium-level reward. We then select the second perturbation that has the biggest reward gain when the first perturbation step is applied. The intermediate candidate is then constructed by applying the first and second perturbation steps on the original candidate segmentation. For BoundRL w/o select, we incorporate intermediate candidates for all input texts where the intermediate candidate has a positive reward gain over the original generated candidate segmentation, rather than restricting replacement to the top-k cases. For BoundRL w/o middle, we construct intermediate candidates by perturbing a randomly sampled candidate segmentation instead of the one with the medium-level reward. Other design choices for these ablated versions are the same as those for BoundRL for a fair comparison.

A.9 Evaluation of BoundRL on WikiSection

In addition to the StructSeg, we also evaluate BoundRL using Qwen3-1.7b on WikiSection-city dataset. We compare BoundRL with the same set of baselines as in Sec.6.1. The results are shown in Tab. 8.

As shown in Tab. 8, all methods perform almost the same while BoundRL slightly outperforms all other baselines. The smaller performance gap on WikiSection-city compared with StructSeg is mainly because WikiSection-city is much simpler in terms of structure. Unlike prompts in StructSeg, input texts in WikiSection-city only

contains the plain text and the ground-truth segmentation is on the sentence level instead of token level. However, even in such cases, BoundRL still slightly outperforms other baselines.

Segment the following prompt into different categories and referring to their descriptions:
<categories>
* <option>instruction</option>: Including 1/ profile/role that the model is acting as; 2/ Core intent of the prompt; 3/ workflow or steps and processes the model should follow to complete the task; 4/ restrictions on what the model must adhere to when generating responses
* <option>context</option>: Background information and context that the model needs to refer to. This can include 1/ background or supplementary input that helps set the stage for the task but is not the primary focus. 2/ Knowledge input - The core content that the prompt directly processes or manipulates; 3/ Metadata/Short Phrases - Brief inputs or settings that define specific parameters or goals for the task.
* <option>question</option>: Queries or questions provided specifically by users. The questions that are part of the template should not be labeled as question but as instruction.
* <option>examples</option>: Providing the AI model with concrete examples of the desired input-output pattern before asking it to perform a similar task. These examples demonstrate the expected format, style, and reasoning pattern, helping the model understand and replicate the desired behavior. A example must contain both concrete input and output. If either input or output is missing, it should not be labeled as example but be labeled as instruction or output format. If input or output does not have actual content, it should not be labeled as example.
* <option>output_format</option>: Specific requirement of the type, format, or style of the output, such as the exact json format or function calling language. A general requirement like 'the output should be in json format' should not be labeled as output_format.
</categories>
<notes>
* Do not separate consecutive prompt segments if they belong to the same category. Make them into one component.
* If unclear, or you are looking at less meaningful text pieces between different components, label it as instruction
* Must keep the text of each component exactly the same as the original prompt.
* Your response between <segmentation_annotation> </segmentation_annotation> must be parsable by Python's ast.literal_eval(). Avoid use single quotes within text of each component.
</notes>
<example>
<example_prompt>
{example_prompt}
</example_prompt>
<example_segmentation>
{example_segmentation}
</example_segmentation>
</example>
<prompt_to_analyze>
{prompt_to_analyze}
</prompt_to_analyze>
<segmentation_annotation>
[{{
   'relative_order': 0,
   'text': [text of component 1],
   'type': [type of component 1, given categories only],
 }},
...
{{
   'relative_order': N,
   'text': [text of component N],
   'type': [type of component N, given categories only],
 }},
]
</segmentation_annotation>
provide your annotated answer enclosed in <segmentation_annotation></segmentation_annotation> XML tags.

Figure 3: Prompt used by Claude to extract the full text of each segment

Segment the following prompt into different categories and referring to their descriptions:
<categories>
* <option>instruction</option>: Including 1/ profile/role that the model is acting as; 2/ Core intent of the prompt; 3/ workflow or steps and processes the model should follow to complete the task; 4/ restrictions on what the model must adhere to when generating responses
* <option>context</option>: Background information and context that the model needs to refer to. This can include 1/ background or supplementary input that helps set the stage for the task but is not the primary focus. 2/ Knowledge input - The core content that the prompt directly processes or manipulates; 3/ Metadata/Short Phrases - Brief inputs or settings that define specific parameters or goals for the task.
* <option>question</option>: Queries or questions provided specifically by users. The questions that are part of the template should not be labeled as question but as instruction.
* <option>examples</option>: Providing the AI model with concrete examples of the desired input-output pattern before asking it to perform a similar task. These examples demonstrate the expected format, style, and reasoning pattern, helping the model understand and replicate the desired behavior. A example must contain both concrete input and output. If either input or output is missing, it should not be labeled as example but be labeled as . If input or output does not have actual content, it should not be labeled as example.
* <option>output_format</option>: Specific requirement of the type, format, or style of the output, such as the exact json format or function calling language. A general requirement like 'the output should be in json format' should not be labeled as output_format.
</categories>
<notes>
* Do not separate consecutive prompt segments if they belong to the same category. Make them into one segment.
* If unclear, or you are looking at less meaningful text pieces between different segments, label it as instruction
* Only output starting words (less than 10 words) of each segment instead of outputting the whole segment
* Must keep the starting words of each segment exactly the same as the corresponding words of the original prompt.
* The extracted segments should be in the order as they are in the original prompt.
* Your response between <segmentation_annotation> </segmentation_annotation> must be parsable by Python's ast.literal_eval(). Avoid use single quotes within text of each component.
</notes>
<example>
<example_prompt>
{example_prompt}
</example_prompt>
<example_segmentation>
{example_segmentation}
</example_segmentation>
</example>
<prompt_to_analyze>
{prompt_to_analyze}
</prompt_to_analyze>
<segmentation_annotation>
[{{
  'relative_order': 0,
  'text': [starting words of segment 1],
  'type': [type of segment 1, given categories only],
 }},
...
{{
  'relative_order': N,
  'text': [starting words of segment N],
  'type': [type of segment N, given categories only],
 }}]
</segmentation_annotation>
provide your annotated answer enclosed in <segmentation_annotation></segmentation_annotation> XML tags.

Figure 4: Prompt used by Claude to extract the starting tokens of each segment

You are requested to segment a prompt into following categories:

(1) instruction: A guidance on how to process and respond to queries.

(2) context: Background information and context that the model needs to refer to.

(3) question: Queries or questions provided specifically by users.

(4) examples: Examples of what the input / output should look like.

(5) output_format: The type, format, or style of the output.

You should output the segments of a prompt in the form of [category_1]start_tokens_1%<separator>%\n[category_2]start_tokens_2...\n[category_N]start_tokens_N, where 'category_1' denotes the predicted category for segment 1 and 'start_tokens_1' denotes the start tokens of segment 1.

Given the following LLM prompt, please follow the instruction above to complete the data field extraction task.

Figure 5: Meta instruction used by BoundRL to output starting tokens of each segment.

You are requested to segment a prompt into following categories:

(1) instruction: A guidance on how to process and respond to queries.

(2) context: Background information and context that the model needs to refer to.

(3) question: Queries or questions provided specifically by users.

(4) examples: Examples of what the input / output should look like.

(5) output_format: The type, format, or style of the output.

You should output the segments of a prompt in the form of [category_1]end_tokens_1%<separator>%\n[category_2]end_tokens_2...\n[category_N]end_tokens_N, where 'category_1' denotes the predicted category for segment 1 and 'end_tokens_1' denotes the end tokens of segment 1.

Given the following LLM prompt, please follow the instruction above to complete the data field extraction task.

Figure 6: Meta instruction used by BoundRL to output ending tokens of each segment.

You are requested to segment a prompt into following categories:
(1) instruction: A guidance on how to process and respond to queries.
(2) context: Background information and context that the model needs to refer to.
(3) question: Queries or questions provided specifically by users.
(4) examples: Examples of what the input / output should look like.
(5) output_format: The type, format, or style of the output.

You should output the segments of a prompt in the form of [category_1]start_tokens_1%<separator>%end_tokens_1[/category_1]\n...\n[category_N]start_tokens_N%<separator>%end_tokens_N[/category_N], where 'category_1' denotes the predicted category for segment 1, 'start_tokens_1' denotes the start tokens of segment 1, and 'end_tokens_1' denotes the end tokens of segment 1.

Given the following LLM prompt, please follow the instruction above to complete the data field extraction task.

Figure 7: Meta instruction used by BoundRL to output both starting and ending tokens of each segment.

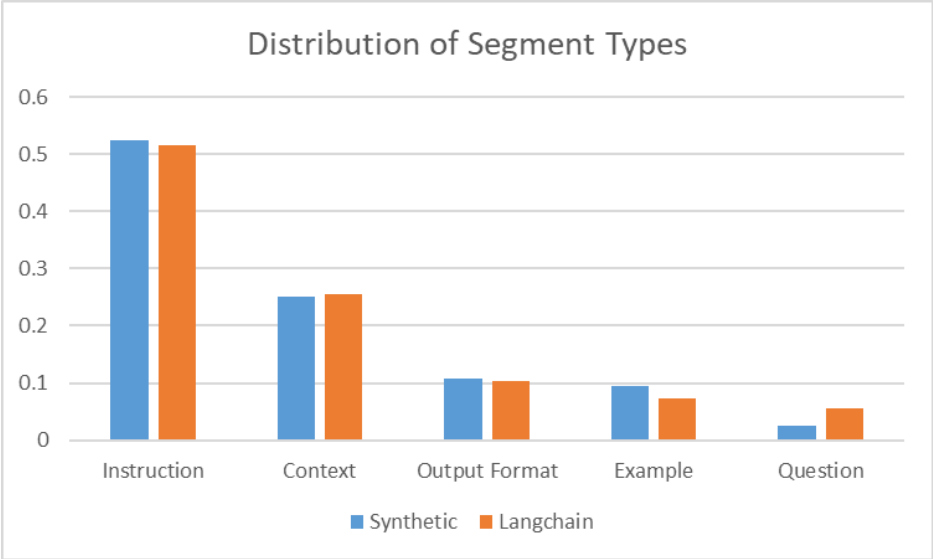


Figure 8: Distribution of segment labels across our dataset showing the proportion of each label type in both synthetic prompts (Synthetic) and real-world prompts (Langchain).

Prompt: Your a trivia expert. Answer questions precise and quick. Use this format:

Q: [Question]
A: [Answer]
Confidence: [0-100%]
Fun Fact: [Related interesting tidbit]

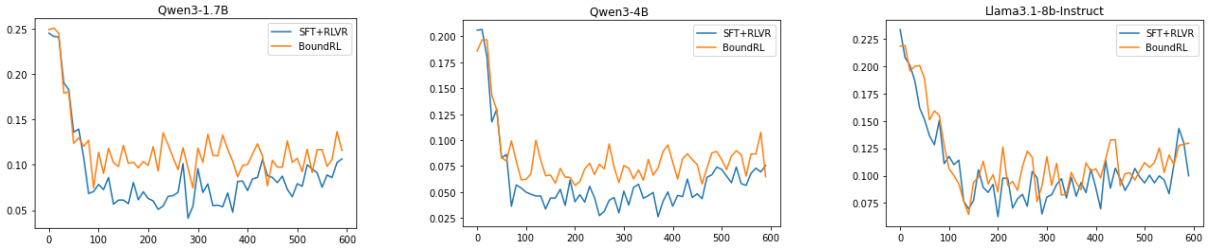
Now, answer these trivia questions:

1. What's the capital of France?
2. Who painted the Mona Lisa?
3. In what year did World War II end?
4. What's the largest planet in our solar system?
5. Who wrote the play "Romeo and Juliet"?

Provide answers for all 5 questions using the specified format. Don't skip any parts of the format. Be concise but accurate in your responses.

Start:[instruction]Your a trivia expert. Answer questions %<separator>% [question]1. What's the capital of France? 2. Who %<separator>% [instruction]Provide answers for all 5 questions %<separator>%	**End:**[instruction]answer these trivia questions: %<separator>% [question]Who wrote the play "Romeo and Juliet"? %<separator>% [instruction]but accurate in your responses. %<separator>%	**Start+End:**[instruction]Your a trivia expert. Answer questions %<separator>% answer these trivia questions:[/instruction] [question]1. What's the capital of France? 2. Who %<separator>% Who wrote the play "Romeo and Juliet"?[/question] [instruction]Provide answers for all 5 questions %<separator>% but accurate in your responses.[/instruction]

Figure 9: An example text and expected outputs for different output patterns.



(a) Qwen3-1.7b (b) Qwen3-4b (c) Llama3.1-8b-Instruct

Figure 10: The standard deviation of rewards during training for BoundRL and SFT+RLVR. Intermediate candidates help BoundRL mitigate the entropy collapse issue of RLVR.

Json Snippet: Parse the JSON data provided below, which contains customer reviews ……
Here's the JSON data containing customer reviews:
{
"reviews": [
{
"product_id": "SH001",
"product_name": "SmartHub 2000",
"category": "Hub",
"rating": 4,
"review_text": "Great central hub for all my smart devices.",
},
[truncated…]
]
}
To assist you in your analysis, you can use the following function calls:
1. analyze_sentiment(text: str) -> dict:
This function performs sentiment analysis on the given text and returns a dictionary with positive, negative, and neutral scores.

Figure 11: Sentences split by the $\text{Oracle}_{\text{sent}}$'s sentence tokenizer. The first sentence is in blue. The second sentence is in green. The sentence tokenizer cannot effectively handle structured elements that do not follow conventional sentence boundaries.

Factor	Values
task type	'named entity recognition', 'current events knowledge', 'document similarity comparison', 'bug detection', 'grammar and spell checking', 'code refactoring', 'parameter extraction', 'software development', 'webhook handling', 'clinical note summarization', 'table relationship inference', 'trivia answering', 'anything you can think that a user might need help with', 'web development', 'general productivity assistant', 'spam detection', 'scientific concept explanation', 'mathematical problem solving', 'email thread summarization', 'multi-option reasoning', 'genre classification', 'research paper abstracting', 'classification', 'database schema understanding', 'general coding assistance', 'text simplification', 'keyword extraction', 'news categorization', 'multi-API orchestration', 'fact-checking', 'sentiment analysis', 'citation finding', 'meeting minutes generation', 'news article summarization', 'fact verification', 'legal document summarization', 'complex query generation', 'toxicity detection', 'factoid QA', 'technical domain QA', 'semantic search', 'geographic knowledge QA', 'logical reasoning tasks', 'text generation', 'text style transfer', 'function calling', 'contextual recommendations', 'language detection', 'api authentication', 'evidence extraction', 'table-based QA', 'data filtering', 'code review', 'biographical information retrieval', 'open_book_qa (RAG), where a document is provided and a quesiton must be answered, but don\'t explicitly mention "open book qa"', 'dialogue summarization', 'topic classification', 'code explanation', 'code security enhancements', 'data type validation', 'anomaly detection in text', 'common sense reasoning', 'code generation', 'metaphor generation and interpretation', 'document type classification', 'ai coding assistant', 'poetry and song lyrics generation', 'customer feedback summarization', 'general programming ai assistance', 'code completion', 'chart/graph interpretation', 'automated essay scoring', 'paraphrasing', 'closed_book_qa, but don\'t explicitly mention "closed book qa"', 'multi-hop reasoning', 'input sanitzation', 'content moderation', 'SQL query optimization', 'api endpoint selection', 'context-dependent reasoning', 'error handling', 'intent classification', 'summarization', 'emotion classification', 'cross-document QA', 'multi-document summarization', 'text2sql', 'text completion', 'code summarization', 'reading comprehension', 'historical fact retrieval', 'video transcript summarization'
writing styles	'to contain several noticeable grammatical errorsin direct and curt way', 'to contain several noticeable grammatical errors', 'in direct and curt way', 'to have lots of typos', 'in a well-formed style'
format type	'a mixture of markdown and formatting seen in the example prompts provided above', 'a mixture of a formatting structure of your choice and section subtitles', 'a mixture of JSON or nested JSON and other', 'markdown', 'a mixture of JSON or nested JSON and markdown', 'YAML-style formatting', 'a mixture of a formatting structure of your choice and other', 'a mixture of markdown and section headers', 'a mixture of formatting seen in the example prompts provided above and section subtitles', 'a mixture of other and markdown', 'a mixture of XML tags and coding', 'a mixture of coding and XML tags', 'a mixture of section subtitles and a formatting structure of your choice', 'a mixture of JSON or nested JSON and coding', 'a mixture of a formatting structure of your choice and markdown', 'a mixture of section subtitles and coding', 'a mixture of formatting seen in the example prompts provided above and JSON or nested JSON', 'a mixture of XML tags and section subtitles', 'a mixture of XML tags and formatting seen in the example prompts provided above', 'a mixture of section headers and formatting seen in the example prompts provided above', 'a mixture of a formatting structure of your choice and coding', 'pseudo-code', 'a mixture of coding and a formatting structure of your choice', 'a mixture of coding and JSON or nested JSON', 'a mixture of markdown and coding', 'a mixture of other and JSON or nested JSON', 'a mixture of XML tags and a formatting structure of your choice', 'section subtitles', 'a mixture of other and a formatting structure of your choice', 'a mixture of section headers and section subtitles', 'a mixture of other and coding', 'a mixture of section subtitles and markdown', 'a mixture of section subtitles and other', 'a mixture of coding and section headers', 'a mixture of section headers and a formatting structure of your choice', 'a mixture of section headers and coding', 'chain-of-thought styling', 'capital letters to highlight important details', 'a mixture of other and section subtitles', 'a mixture of XML tags and JSON or nested JSON', 'XML tags', 'a mixture of coding and other', 'a mixture of other and formatting seen in the example prompts provided above', 'a mixture of formatting seen in the example prompts provided above and section headers', 'a mixture of a formatting structure of your choice and XML tags', 'a mixture of formatting seen in the example prompts provided above and other', 'a formatting structure of your choice', 'a mixture of section headers and XML tags', 'a mixture of XML tags and markdown', 'a mixture of JSON or nested JSON and a formatting structure of your choice', 'a mixture of markdown and a formatting structure of your choice', 'JSON or nested JSON', 'a mixture of a formatting structure of your choice and section headers', 'a mixture of JSON or nested JSON and section headers', 'a mixture of coding and formatting seen in the example prompts provided above', 'a mixture of JSON or nested JSON and formatting seen in the example prompts provided above', 'a mixture of section subtitles and section headers', 'a mixture of JSON or nested JSON and XML tags', 'a mixture of other and XML tags', 'a mixture of XML tags and other', 'a mixture of section headers and JSON or nested JSON', 'a mixture of markdown and JSON or nested JSON', 'a mixture of a formatting structure of your choice and formatting seen in the example prompts provided above', 'a mixture of section subtitles and XML tags', 'table-based formatting', 'a mixture of a formatting structure of your choice and JSON or nested JSON', 'a mixture of section headers and other', 'a mixture of formatting seen in the example prompts provided above and markdown', 'a mixture of formatting seen in the example prompts provided above and a formatting structure of your choice', 'a mixture of section headers and markdown', 'a mixture of markdown and section subtitles', 'a mixture of markdown and XML tags', 'tree-style hierarchical formatting', 'a mixture of section subtitles and formatting seen in the example prompts provided above', 'a mixture of coding and markdown', 'a mixture of section subtitles and JSON or nested JSON', 'coding', 'a mixture of other and section headers', 'a mixture of coding and section subtitles', 'a mixture of markdown and other', 'formatting seen in the example prompts provided above', 'a mixture of XML tags and section headers', 'a mixture of formatting seen in the example prompts provided above and XML tags', 'section headers', 'a mixture of JSON or nested JSON and section subtitles', 'a mixture of formatting seen in the example prompts provided above and coding'
prompt length	'less than 150 words', '150 to 500 words', '500 to 1000 words', 'around 1000 words', '1000 to 2000 words'
level of detail	'basic level of detail, meaning it can just give minimal descriptions of things', 'moderate level of detail, meaning it goes a bit in-depth into things', 'detailed, meaning you should describe things thoroughly and do not give short names or descriptions', 'very detailed, meaning everything is described very in-depth and production-level detail is included', 'extremely technically detailed, meaning as many specific details should be present as possible, including technical jargon, production-level of context, and complicated descriptions'

Table 7: Full list of factors and corresponding potential values used for generating of synthetic prompts.

	ρ_{rec}	EM	P_k	F1_{lab}	F1_{char}	Avg.
SFT	0.993	0.701	0.066	0.817	0.816	0.852
SFT w/2epoches	0.993	**0.713**	0.064	0.819	0.825	0.857
SFT+RLVR	0.994	0.711	0.063	**0.826**	0.824	0.859
SFT+RLVR w/high temp.	**0.995**	0.709	0.064	0.824	0.824	0.857
RL-PLUS	0.993	0.709	0.063	0.819	0.825	0.857
BoundRL	0.994	0.712	**0.062**	0.825	**0.827**	**0.859**

Table 8: Evaluation of BoundRL using Qwen3-1.7b on WikiSection-city dataset. The best-performing method highlighted in **bold**.

Prompt: You are a supervisor tasked with managing a conversation between the following workers: {members}. Given the following user request, respond with the worker to act next. Here are the descriptions for each worker:
{worker_desc}
Workers must complete a task and report their results and status.
NEVER reply with FINISH if AI provides no response.
NOTE: Stop if messages are repeating after three attempts.
Given the conversation above, who should act next?
Or should we FINISH?
Select one of: {options}

Annotation: [{'text': 'You are a supervisor tasked with managing a conversation between the following workers:', 'type': 'instruction'}, {'text': '{members}', 'type': 'context'}, {'text': '. Given the following user request, respond with the worker to act next. Here are the descriptions for each worker:', 'type': 'instruction'}, {'text': '{worker_desc}', 'type': 'context'}, {'text': 'Workers must complete a task and report their results and status.\n NEVER reply with FINISH if AI provides no response.\n NOTE: Stop if messages are repeating after three attempts.\n\n Given the conversation above, who should act next?\n Or should we FINISH? \n Select one of:', 'type': 'instruction'}, {'text': '{options}', 'type': 'context'}]

SFT w/2epochs

Raw output: [instruction]You are a supervisor tasked with %<separator>% [context]{members} %<separator>% [instruction]Given the following user request, %<separator>% [context]{worker_desc} %<separator>% [instruction]Workers must complete a task and %<separator>% [context]Or should we %<separator>% [instruction]Select one of: %<separator>% [context]{options} %<separator>%

Reconstucted Segment: [{'text': '', 'label': 'instruction'}, {'text': '', 'label': 'context'}, {'text': 'You are a supervisor tasked with managing a conversation between the following workers:', 'label': 'instruction'}, {'text': '{worker_desc}', 'label': 'context'}, {'text': 'Workers must complete a task and report their results and status.\n NEVER reply with FINISH if AI provides no response.\n NOTE: Stop if messages are repeating after three attempts.\n\n Given the conversation above, who should act next?', 'label': 'instruction'}, {'text': 'Or should we FINISH?', 'label': 'context'}, {'text': 'Select one of:', 'label': 'instruction'}, {'text': '{options}', 'label': 'context'}]

SFT+RLVR

Raw output: [instruction]You are a supervisor tasked with %<separator>% [context]{members} %<separator>% [instruction]Given the following user %<separator>% [context]{worker_desc} %<separator>% [instruction]Workers must complete a task and %<separator>% [question]Or should we %<separator>% [instruction]Select one of: %<separator>% [context]{options} %<separator>%

Reconstucted Segment: [{'text': 'You are a supervisor tasked with managing a conversation between the following workers:', 'label': 'instruction'}, {'text': '{members}.', 'label': 'context'}, {'text': 'Given the following user request, respond with the worker to act next. Here are the descriptions for each worker:', 'label': 'instruction'}, {'text': '{worker_desc}', 'label': 'context'}, {'text': 'Workers must complete a task and report their results and status.\n NEVER reply with FINISH if AI provides no response.\n NOTE: Stop if messages are repeating after three attempts.\n\n Given the conversation above, who should act next?', 'label': 'instruction'}, {'text': 'Or should we FINISH?', 'label': 'question'}, {'text': 'Select one of:', 'label': 'instruction'}, {'text': '{options}', 'label': 'context'}]

RL-PLUS

Raw output: [instruction]You are a supervisor tasked with %<separator>% [context]{members} %<separator>% [instruction]Given the following user %<separator>% [context]{worker_desc} %<separator>% [instruction]Workers must complete a task and %<separator>% [context] %<separator>% [instruction]Or should we %<separator>% [context]{options} %<separator>%

Reconstucted Segment: [{'text': 'You are a supervisor tasked with managing a conversation between the following workers:', 'label': 'instruction'}, {'text': '{members}.', 'label': 'context'}, {'text': 'Given the following user request, respond with the worker to act next. Here are the descriptions for each worker:', 'label': 'instruction'}, {'text': '{worker_desc}', 'label': 'context'}, {'text': 'Workers must complete a task and report their results and status.\n NEVER reply with FINISH if AI provides no response.\n NOTE: Stop if messages are repeating after three attempts.\n\n Given the conversation above, who should act next?', 'label': 'instruction'}, {'text': 'Or should we FINISH? \n Select one of:', 'label': 'instruction'}, {'text': '{options}', 'label': 'context'}]

BoundRL

Raw output: [instruction]You are a supervisor tasked %<separator>% [context]{members} %<separator>% [instruction]Given the following user %<separator>% [context]{worker_desc} %<separator>% [instruction]Workers must complete a task and %<separator>% [context]{options} %<separator>%

Reconstucted Segment:[{'text': 'You are a supervisor tasked with managing a conversation between the following workers:', 'label': 'instruction'}, {'text': '{members}.', 'label': 'context'}, {'text': 'Given the following user request, respond with the worker to act next. Here are the descriptions for each worker:', 'label': 'instruction'}, {'text': '{worker_desc}', 'label': 'context'}, {'text': 'Workers must complete a task and report their results and status.\n NEVER reply with FINISH if AI provides no response.\n NOTE: Stop if messages are repeating after three attempts.\n\n Given the conversation above, who should act next?\n Or should we FINISH? \n Select one of:', 'label': 'instruction'}, {'text': '{options}', 'label': 'context'}]

Figure 12: Qualitative examples of BoundRL and other baselines.