DD-GEPA: Prompt Optimization for Dialogue Disentanglement Focusing on Task Instruction and Utterance Representation

Anonymous ACL submission

Abstract

Multi-party chat often contains interleaved dialogues because multiple participants can discuss different topics at the same time. Dialogue disentanglement addresses this problem by separating an entangled utterance sequence into coherent dialogues. While large language models (LLMs) are promising for this task, they still struggle with dialogue disentanglement and achieve low accuracy. This paper proposes an automatic prompt optimization for LLM based dialogue disentanglement. We decompose the prompt into three components: task instruction, utterance representation, and output instruction, and optimize them using GEPA, an optimization method for compound AI systems. Experiments on benchmark datasets show that the optimized prompts improve dialogue disentanglement accuracy over the original prompts and can surpass hand crafted prompts.

1 Introduction

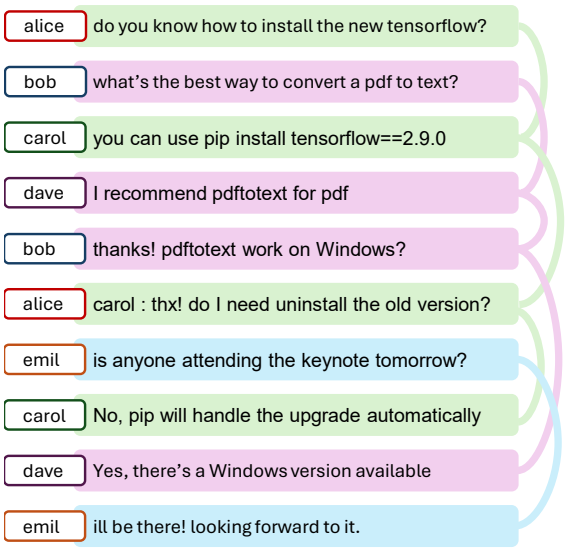

Figure 1: An example of dialogue disentanglement, where the resulting dialogues are shown by color.

In multi-party chat, such as Slack and Internet Relay Chat(IRC), multiple dialogues are entangled. Because many users send messages simultaneously, multiple dialogues can proceed in parallel and become entangled. Such entanglement can also lead to failures in automatic analysis, because utterances from unrelated dialogues can be mistaken for relevant context. To address this problem, dialogue disentanglement was proposed (Elsner and Charniak, 2010). As shown in Figure 1, dialogue disentanglement is the task of partitioning entangled utterance sequence into coherent dialogue clusters whose utterances are connected by reply-to relations. This task is an important preprocessing for downstream applications such as dialogue state tracking (Ouyang et al., 2020) and response generation (Cai et al., 2022). A common approach to disentangle is to identify reply-to relations between utterances (Elsner and Charniak, 2010). For each utterance, we determine whether the utterance is (1) begginning of a new dialogue or (2) reply to a previous utterance, and then construct the each disentangled dialogues by linking these reply-to relations.

Large language models (LLMs) have strong contextual reasoning capabilities, suggesting that they can be effectively applied to dialogue disentanglement. However, the first attempt to apply LLMs to this task achieved very low accuracy and fell far short of existing non LLM methods (Li et al., 2025). Later work reported that prompt design can improve the accuracy of LLM based dialogue disentanglement. In particular, introducing dialogue-level assignment and subsequent context achieved accuracy that surpassed existing methods, demonstrating the potential of LLMs for dialogue disentanglement (Takada and Mori, 2026). However, the effectiveness of dialogue-level assignment and subsequent context varies across LLMs, and these components can even reduce accuracy. Thus, there is still no generally effective

method for improving LLM based dialogue disentanglement. Because chat data may contain personal information, accurate disentanglement with open-weight LLMs is desirable. However, open-weight LLMs with around 30B parameters still fall far short of conventional non LLM methods, leaving substantial room for improvement.

Dialogue disentanglement is a challenging task even for humans; prior annotation studies have shown substantial variation in judgments among annotators (Kummerfeld et al., 2019). This makes it difficult to manually write a clear task instruction for dialogue disentanglement. Moreover, LLM performance is highly sensitive to how information and output formats are specified in the prompt (Sclar et al., 2024; Tam et al., 2024). The same issue is pronounced in dialogue disentanglement: as shown later in Section 5, especially Table 1, accuracy changes substantially depending on the utterance representation and output instruction. These observations indicate that prompt design must account not only for the task instruction but also for the output instruction. Because jointly designing and validating these components by hand is difficult, this study proposes an automatic prompt optimization method for dialogue disentanglement. We decompose each prompt into the following three components and optimize them separately.

Task instruction: This component defines dialogue disentanglement, explains the task, and specifies what evidence should be used for the assignment decision.

Utterance representation: This component presents each utterance as a combination of utterance number, timestamp, speaker ID, and message content. It specifies how this information is formatted for the LLM, for example by using JSON or adding explicit labels to each field.

Output instruction: This component specifies how to output the result of dialogue disentanglement.

Our optimization target is a prompt composed of multiple components. We therefore propose a method based on GEPA (Agrawal et al., 2026), which has achieved state-of-the-art results in automatic optimization for compound AI systems. The main contribution of this work is an automatic method for optimizing dialogue disentanglement prompts, whose design is difficult to specify manually. Experiments show that the optimized prompt improves accuracy over the initial prompts and the manually designed prompt of Takada and Mori (2026).

2 Related Work

2.1 Dialogue Disentanglement

Dialogue disentanglement has been formulated as a two stage problem (Elsner and Charniak, 2010; Li et al., 2020; Ma et al., 2021). First, a classifier assigns scores to utterance pairs within a specified context window to estimate the strength of their reply-to relations. Second, a clustering algorithm aggregates the predicted local links to construct the global dialogue structure. Early methods estimated reply-to relations from manually designed features, including lexical overlap, time gaps, and explicit user mentions (Elsner and Charniak, 2008, 2010). The release of a large annotated corpus (Kummerfeld et al., 2019) encouraged the development of end to end neural models, and the use of fine tuned PLMs such as BERT further improved accuracy by utilizing richer contextual information (Li et al., 2020). However, these methods mainly model the utterance sequence itself and make limited use of dialogue specific information. Subsequent work incorporated speaker information and user mentions (Ma et al., 2021). DiHRL (Li et al., 2025) further improved performance by integrating hierarchical learning losses, an easy-first decoding algorithm, and global conversational features.

Although LLMs have achieved strong results across natural language processing tasks, the ability of dialogue disentanglement remains underexplored. The first attempt (Li et al., 2025) substantially underperformed conventional non LLM methods. Its task instruction was minimal, asking the LLMs only to identify an utterance in a reply-to relation and output its index, with a short hint that nearby utterances are likely to be related. Its utterance representation and output instruction were also examined only through simple examples. To address this issue, Takada and Mori (2026) proposed an LLM based method that substantially outperformed prior methods by improving the utterance representation and output instruction. Their approach represents the utterance sequence in JSON format and introduces dialogue-level assignment (DLA), which presents dialogue structures inferred from previous assignments, and subsequent context (SC), which adds utterances following the target utterance as auxiliary evidence.

However, while these techniques achieve high accuracy with some proprietary LLMs, performance drops sharply for around 30B open-weight LLMs. High accuracy disentangling with smaller open-weight LLMs is especially important when handling chat data that may contain personal information. This work addresses that gap by improving the accuracy of LLM based dialogue disentanglement.

One possible path to higher accuracy is to provide a more informative task instruction. Such an instruction should clearly define the criteria for assigning each target utterance. Human annotation guidelines could serve as a natural starting point. However, although many dialogue disentanglement annotations have been conducted (Kummerfeld et al., 2019; Chatterjee et al., 2020), detailed guidelines have not been established. The only available annotation guideline is that of Kummerfeld et al. (2019), which is brief and requires reconciliation among multiple annotators. Even with this guideline, inter annotator agreement was below a kappa coefficient of 0.75, indicating substantial variation in judgment. This variation stems from the difficulty of dialogue disentanglement annotation. For each target utterance, annotators must choose parent utterances from many preceding candidates. Multi-party chat also contains diverse topics, participant configurations, and dialogue situations. These factors make it difficult to write a task instruction that enables accurate LLM based dialogue disentanglement. Prompt design is harder still because accuracy also depends on the utterance representation and output instruction. We therefore address prompt design through automatic prompt optimization.

2.2 Automatic Prompt Optimization

Effective use of LLMs requires appropriate prompt design, but manual design involves extensive trial and error and does not guarantee an optimal prompt. Automatic prompt optimization has therefore become an active research area. Early methods include Prompt Tuning (Lester et al., 2021) and Prefix Tuning (Li and Liang, 2021). Instead of searching for natural language instructions, these methods optimize learnable continuous vectors attached to the input. The LLM parameters remain fixed, and only the added vectors are trained. Although this is more efficient than full fine tuning, the resulting prompts are not human readable. They are also model specific, which makes them difficult to transfer across LLMs.

In contrast, APE (Zhou et al., 2023) and OPRO (Yang et al., 2024) optimize natural language prompts. They use an LLM as the optimizer to generate and refine candidate prompts. Because the prompts are written in natural language, humans can inspect and reuse them more easily across LLMs. EvoPrompt (Guo et al., 2024) incorporates evolutionary search into prompt optimization, allowing it to explore a broader space of candidate prompts. However, these methods are mainly designed for single prompt, single task settings. They are less suited to compound AI systems in which multiple agents or reasoning steps interact. To address this limitation, MIPROv2 (Opsahl-Ong et al., 2024) extends prompt optimization to compound systems using Bayesian optimization and bootstrap search. GEPA (Agrawal et al., 2026) further improves compound system optimization by using natural language reflection on execution trajectories and evolutionary Pareto based selection. We choose GEPA as our optimization method for the following three reasons.

First, dialogue disentanglement prompts involve multiple factors, and GEPA is suitable for optimizing such multi component systems. Second, GEPA optimizes natural language prompts, which makes the resulting prompts interpretable and more portable when the underlying LLM changes. Third, GEPA has achieved state-of-the-art performance. However, GEPA cannot be applied directly to dialogue disentanglement. In its original form, GEPA optimizes only the task instruction and does not handle the utterance representation or output instruction. We therefore adapt GEPA and propose a new method that optimizes these components together for dialogue disentanglement.

3 LLM Based Dialogue Disentanglement

3.1 Task Formulation

We define the input conversation $C = (u_1, u_2, \ldots, u_n)$ as the full utterance sequence observed in the chat. Each utterance u_i is represented as $u_i = (t_i, s_i, m_i)$, where t_i is the timestamp, s_i is the speaker ID, and m_i is the message content. The desired output is a set of dialogues $D = \{d_1, d_2, \ldots, d_p\}$, where each dialogue d_j is a cluster of utterances that are linked by reply-to relations. The task is to partition C

into the dialogue set D. The partition must satisfy two conditions. First, it must cover all utterances: $C = \bigcup_{d_j \in D} d_j$. Second, the dialogue clusters must be mutually exclusive: $\forall j \neq k, d_j \cap d_k = \emptyset$.

3.2 LLM Based Disentanglement Method

For each utterance, we ask the LLM to predict its reply-to relation. We select one utterance as the target utterance u_{target} and provide it to the LLM together with the utterances that precede it. The LLM determines whether u_{target} starts a new dialogue or responds to a previous utterance. If it responds to a previous utterance, the LLM selects one preceding utterance as the most appropriate parent utterance.

After predicting reply-to relations for all utterances in C, we construct dialogue clusters in chronological order. If u_{target} is predicted to start a new dialogue, we create a new cluster containing only u_{target}. If u_{target} is predicted to reply to a previous utterance, we add it to the cluster that contains the predicted parent utterance. Repeating this process for all utterances yields the set of dialogues D. This algorithm follows conventional non LLM methods (Elsner and Charniak, 2010); in this study, we replace the classifier with an LLM.

In addition to this basic framework, Takada and Mori (2026) proposed two extensions. Dialogue-Level Assignment (DLA) changes the assignment target from individual previous utterances to previously established dialogues. In DLA, the previous context is represented by dialogue structures inferred from earlier assignments. Subsequent Context (SC) adds utterances after u_{target} as auxiliary evidence. However, the contribution of DLA and SC differs across LLMs, and these methods can reduce accuracy in some cases. Since this study focuses on prompt optimization itself, we use the basic framework excluding DLA and SC.

3.3 Scoring Assignment Decisions

GEPA optimizes prompts by evaluating each LLM output. We therefore define a binary score for the LLM's decision in our dialogue disentanglement task. Let d_{gold} denote the gold dialogue cluster to which the target utterance u_{target} should be assigned.

The LLM decision is evaluated in two cases. If the LLM predicts that u_{target} starts a new dialogue, the decision is correct when u_{target} is the first utterance in d_{gold}. If the LLM predicts that u_{target} replies to a previous utterance, the decision is correct when the predicted parent utterance appears before u_{target} and belongs to d_{gold}. A correct decision receives a score of 1, and any other decision receives a score of 0.

4 Automatic Prompt Optimization

4.1 DD-GEPA

We propose DD-GEPA, which adapts GEPA to dialogue disentanglement prompt optimization. As described in Section 1, we decompose a dialogue disentanglement prompt into three components: task instruction, utterance representation, and output instruction. We refer to each component as a "module" and to a dialogue disentanglement system that consists of the three modules as a "program". DD-GEPA maintains a pool of candidate programs and repeatedly updates one module in one selected program. When an update improves the program, the new program is added to the candidate pool. Figure 2 illustrates this optimization process.

The optimization starts with predefined initial programs. Each program is evaluated on 300 validation instances using the binary decision score. DD-GEPA then selects a Pareto based candidate for improvement in two steps. First, DD-GEPA removes programs dominated by another candidate that solves the same validation instances and achieves a higher total score. Second, one of the remaining programs is sampled with probability proportional to its total score. After the program is selected, the module to update is chosen in round robin order. This selection procedure is unchanged from the original GEPA.

New prompts are generated through reflection on execution traces. In the original GEPA framework, how to construct these traces is left to the user. We define them as follows. For each update, we sample minibatch that has three instances from the train set and run the selected program on them. For each instance, we extract the LLM input, output, reasoning text, and decision correctness as the execution trace. For incorrect decisions, we also ask an LLM to generate a failure explanation and append it to the trace.

After collecting the traces, DD-GEPA updates the selected module. The LLM receives traces and returns an updated prompt for the selected module only; the other two modules are kept unchanged. The updated module is then combined with the unchanged modules to form a new program. If

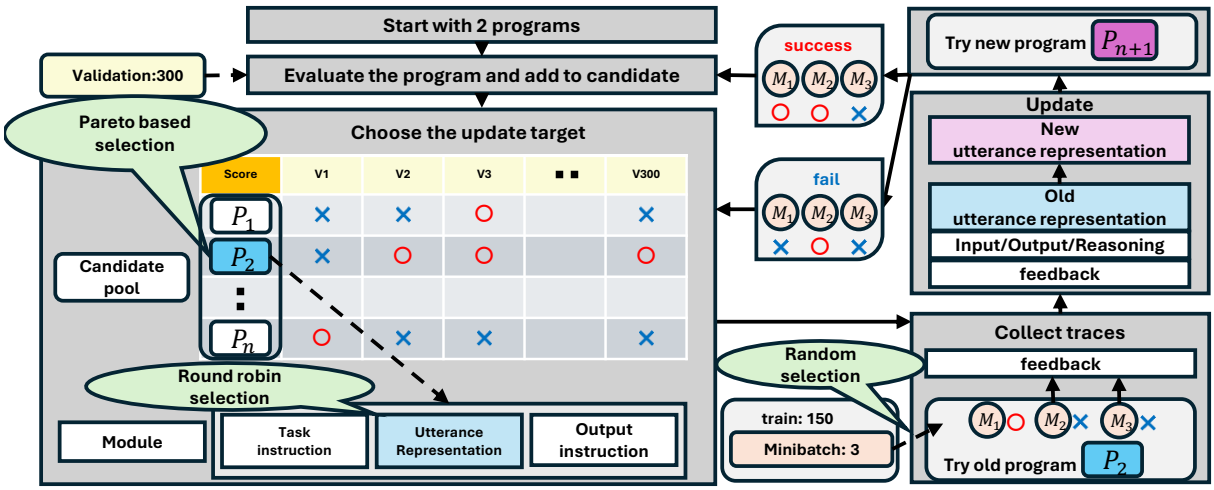

Figure 2: Overview of DD-GEPA for automatic optimization of dialogue disentanglement prompts. A program is selected from the candidate pool using Pareto based selection, and one module in the selected program is chosen for update in round robin order. DD-GEPA samples a minibatch of three training instances, collects execution traces from the selected program, and updates the target module based on the traces. The updated module is incorporated into a new program, which is added to the candidate pool if it improves the task success rate on the minibatch.

the new program performs better than the previous program on the minibatch, the update is accepted. The new program is added to the candidate pool. We then evaluate it on the 300 validation instances so that it can be considered in later program selection. If the update does not improve performance, the new program is discarded. We repeat this cycle for a fixed number of iterations and select the program with the highest validation score as the optimized program. The prompts used in this study are provided in Appendix A.

For dialogue disentanglement, the original GEPA can optimize the task instruction only. DD-GEPA extends the optimization target to include the utterance representation and output instruction. The following subsections describe how each of the three modules is optimized.

4.2 Task Instruction

For the task instruction module, we add one constraint to the original improvement prompt: the generated prompt must not include the output instruction. Without this constraint, the optimizer may mix output requirements into the task instruction, which would prevent us from improving the modules independently.

4.3 Utterance Representation

When updating the utterance representation, we ask the LLM to propose a new representation that improves the accuracy of dialogue disentanglement decisions. To automate prompt optimization, the proposed representation must be applicable to any utterance sequence and inserted into the prompt without manual formatting. We therefore add an external function that generates conversion code. Given the proposed representation, an LLM writes code that converts an utterance sequence into the proposed format and inserts the converted sequence into the prompt during evaluation.

4.4 Output Instruction

For the output instruction, the LLM proposes a more effective prediction format. Since the evaluator expects a fixed format, predictions in a proposed format must be converted before scoring. We therefore add an external function that generates output interpretation code. Given the proposed output instruction, an LLM writes code that converts predictions into the predefined evaluation format. During evaluation, this code converts each prediction to the evaluation format before scoring.

5 Evaluation Experiments

5.1 Experimental Objectives

We evaluate the proposed method from two perspectives. First, we examine whether prompt optimization improves the dialogue disentanglement accuracy of LLMs by comparing a manually designed prompt with the optimized prompt. Second, we compare an optimized prompt with a smaller open-weight LLM against a manually designed prompt with a proprietary LLM to assess

how much optimization can close the performance gap.

5.2 Dataset

We use the Ubuntu IRC dataset, a standard benchmark for dialogue disentanglement (Kummerfeld et al., 2019). The dataset consists of real chat logs in which multiple participants discuss technical issues related to Ubuntu OS, producing interleaved dialogues. It is divided into train, development, and test splits. The development and test splits are high quality gold data validated by multiple annotators; they contain 2,500 and 5,000 utterances, respectively. We use the development split for prompt optimization and the test split for evaluating performance changes and comparing with prior work.

5.3 Prompt Settings

We evaluate four prompt settings used in this study.

Seed 1: Seed 1 is one of the initial programs used for optimization. Its task instruction is a simple instruction originally written for preliminary experiments in DiHRL (Li et al., 2025). Its utterance representation separates utterance fields only with whitespace. Its output instruction requires the LLM to output a single utterance number: the target utterance number if it starts a new dialogue, or the parent utterance number if it replies to a previous utterance.

Seed 2: Seed 2 is another initial program. It differs from Seed 1 only in the output instruction. Instead of a single utterance number, Seed 2 requires two keys: a Boolean key indicating whether the target utterance starts a new dialogue and a key indicating the parent utterance number.

Baseline: Baseline uses a reproduction of the prompt called Baseline in Takada and Mori (2026). It uses the same task instruction as Seed 1, represents each utterance in labeled JSON format, and uses the same output instruction as Seed 2. We treat Baseline as the strongest manually designed prompt for LLM based dialogue disentanglement in this setting and test whether DD-GEPA can surpass it.

Optimum: Optimum is the optimized program obtained by DD-GEPA. Starting from Seed 1, Optimum is obtained by updating the task instruction, utterance representation, and output instruction once each in this order.

Because Baseline achieves higher assignment accuracy than Seed 1 and Seed 2 (see Table 1), the output instruction using two keys and the JSON utterance representation are likely to contribute to performance improvements. However, when optimization starts only from Seed 1, the JSON representation is often proposed early, whereas the output instruction using two keys is rarely proposed. We therefore include Seed 2 as an additional initial program to make the early stage of optimization more effective.

5.4 Evaluation Metrics

We adopt evaluation metrics that are widely used in dialogue disentanglement and evaluate performance from three perspectives. First, we measure the overall consistency of the predicted clustering using Variation of Information (VI) (Meilă, 2003), Adjusted Rand Index (ARI) (Hubert and Arabie, 1985), Normalized Mutual Information (NMI) (McDaid et al., 2011), One-to-One accuracy (1-1) (Elsner and Charniak, 2010), and Shen-F1 (S-F) (Shen et al., 2006). Second, we use Local_3 to measure local clustering accuracy over windows of three utterances (Elsner and Charniak, 2010). Third, we evaluate exact matches of dialogue clusters using precision (P), recall (R), and F1 score (Kummerfeld et al., 2019).

5.5 Hyperparameters

Except for dataset construction (Section 5.6), all experiments use Qwen3-30B-A3B-Thinking-2507 (Yang et al., 2025) as the LLM. The request parameters differ between (i) reply-to relation prediction and (ii) prompt update. For reply-to relation prediction, we set the temperature to 0 to ensure reproducibility in dialogue disentanglement. For prompt update, we follow GEPA and use temperature 0.6, top-p 0.95, and top-k 20 to encourage diverse improvement proposals.

For reply-to relation prediction, we restrict the candidate parent utterances to the 50 preceding utterances. We use GEPA v0.023. Following Agrawal et al. (2026), we use 150 training instances, 300 validation instances, and a minibatch size of 3 for optimization. We omit the GEPA merge operation because its effectiveness was not clearly supported in the original study.

5.6 Construction of the DD-GEPA Dataset

We construct a dataset of 450 instances for prompt optimization, where each instance is a reply-to re-

Table 1: Success rates under different LLM and prompt settings

LLM	Method	Success rate (%)
Qwen3-30B	Seed1	80.28
Qwen3-30B	Seed2	81.60
Qwen3-30B	Baseline	91.92
GPT-5.2 (OpenAI, 2025)	Baseline	96.12

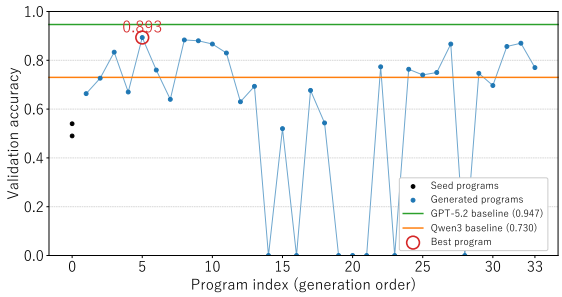

Figure 3: Optimization trajectory of generated programs

lation decision for one target utterance. The instances are drawn from the 2,500 utterances in the development split of the IRC data and are selected to be difficult while still answerable by the LLM. The construction procedure is as follows. First, we collect decision results under different LLM and prompt settings, using Qwen3-30B-A3B as an open-weight LLM and GPT-5.2 as a proprietary LLM, as shown in Table 1. Second, we exclude 63 utterances that all methods fail to classify, treating them as unsuitable for LLM based judgment. This exclusion is necessary because the IRC data contains cases that appear to be annotation errors and cases where the correct reply-to relation lies outside the 50-utterance candidate window. Finally, we treat utterances missed by higher accuracy methods as difficult cases and construct the dataset from them. We split these utterances into training and validation instanses with balanced difficulty.

6 Experimental Results

6.1 Benchmark Evaluation

Table 2 reports performance on the test split of the IRC dataset, which contains 5,000 utterances. The top and middle blocks provide reference results from prior work, whereas the bottom block reports our results under the four prompt settings described in Section 5.3. Optimum improves over the seed prompts and also outperforms Baseline on all evaluation metrics. This result shows that, with Qwen3-30B, DD-GEPA can produce a prompt that is more accurate than the manually designed prompt. However, Optimum still underperforms DiHRL (Li et al., 2025), the strongest non LLM method in the table, and the proprietary LLM results reported by Takada and Mori (2026).

6.2 Limits of Optimization

Figure 3 shows the programs generated during optimization and their validation scores. The best program is produced at the fifth update, and subsequent updates fail to produce a better program. The optimized Qwen3-30B program exceeds the Qwen3-30B Baseline, showing that DD-GEPA can improve on manual prompt design for the same LLMs. However, this optimized open-weight LLM still remains below the GPT-5.2 Baseline, which uses a manually designed prompt with a proprietary LLM. After the fifth update, the validation accuracy stays near 0.88, showing that optimization stops improving after its early gains. Moreover, the updated programs already solve most training instances used for prompt update, leaving few failure cases for reflection. We therefore judge that further optimization is unlikely to improve accuracy in this setting and use the fifth generated program as Optimum.

6.3 Error Analysis

We qualitatively analyze the errors made by Optimum on the IRC test split. The errors fall into five patterns. The first pattern involves greetings and reactions, such as “hi” and “hmm”. When unrelated utterances appear between such short messages from the same speaker, the LLMs may incorrectly split them into different dialogues. The second pattern involves server logs and system messages. Although these utterances do not have reply-to relations, the LLMs sometimes predicts a relation because the message text contains a participant name. The third pattern involves utterances containing Ubuntu specific technical terms, such as “Wine” and “hoary”. In these cases, the LLMs must infer the reply-to relation from topical relatedness, but it often fails to identify the appropriate parent utterance. The fourth pattern involves abstract opinions or comments for which multiple addressees are possible. These cases require the LLMs to infer who the speaker is addressing within a particular dialogue, and errors occur even

Table 2: Comparison of dialogue disentanglement performance. The top block reports prior non LLM methods, the middle block reports prior results with proprietary LLMs, and the bottom block reports our Qwen3-30B results under the four prompt settings: Seed 1, Seed 2, Baseline, and Optimum. Underlining indicates the best score within each block, and boldface indicates the best score overall.

Method		VI	ARI	1-1	NMI	Local_3	S-F	P	R	F1
Elsner (Elsner and Charniak, 2008)		82.10	—	51.40	—	—	—	12.10	21.50	15.50
DiaBERT (Li et al., 2020)		93.20	72.80	79.70	—	—	—	42.10	47.90	44.80
Struct (Ma et al., 2021)		94.60	76.80	84.20	—	—	—	51.80	51.80	51.70
DiHRL (Li et al., 2025)		94.23	81.10	84.20	91.85	95.64	87.50	47.97	49.86	48.90
GPT4.1	DLA+SC(Takada and Mori, 2026)	95.39	82.22	86.34	96.65	95.14	90.55	46.38	48.73	47.53
Gemini2.5pro	DLA+SC(Takada and Mori, 2026)	**97.16**	**92.23**	**90.78**	**97.88**	**97.62**	**92.02**	**58.81**	**63.94**	**61.27**
Qwen3-30B	Seed1	82.97	37.95	56.68	83.83	84.89	63.37	12.03	18.03	14.43
Qwen3-30B	Seed2	83.31	38.98	56.14	84.36	84.68	62.97	11.26	16.34	13.33
Qwen3-30B	Baseline	93.00	70.02	78.46	93.62	94.32	82.81	38.30	40.56	39.40
Qwen3-30B	Optimum	94.12	75.87	82.26	95.51	95.39	84.80	42.22	42.82	42.52

when a participant name is explicitly mentioned. The fifth pattern involves cases where the preceding context is insufficient. Some decisions would become easier if the LLMs could also refer to utterances that follow the target utterance.

7 Discussion

GEPA can reach strong solutions through a long iterative improvement process (Agrawal et al., 2026). However, in our experiments, no further improvement was observed after the best program appeared early in optimization. We consider two possible reasons for this behavior.

First, the optimization process may not provide traces that cover sufficiently diverse cases. Multiparty chat contains wide variation in topics, participant interactions, and conversational style. Considering more traces could help the optimizer handle this diversity. However, the prompt given to the LLM for generating a new prompt already exceeds 20,000 tokens with only three traces. Thus, simply adding more traces is not feasible. A better optimization method may need to incorporate more traces during prompt update while controlling the token cost.

Second, the prompt optimization dataset may still be too easy. Although we constructed the dataset by selecting utterances with high expected difficulty, Figure 3 shows that more than half of the instances are already solved by the seed programs. A larger high quality dialogue disentanglement dataset would make it possible to collect more difficult instances for optimization.

8 Conclusion

This study introduced automatic prompt optimization for LLM based dialogue disentanglement and demonstrated its effectiveness. We decomposed the prompt into task instruction, utterance representation, and output instruction, and optimized these components separately. With a 30B open-weight LLM, the optimized prompt improved dialogue disentanglement accuracy, although its performance remained below that of existing non LLM based methods and unoptimized prompts with proprietary LLMs. We hope that this work will provide a useful basis for optimizing prompts for LLM based dialogue disentanglement.

9 Limitations

This study has several limitations. First, the number of input tokens is bounded, so reply-to relations that fall outside the candidate context window cannot be captured. Second, the dataset used in this study focuses on programming related discussions, and the effectiveness of the proposed method on other domains remains unverified. Third, because the chat data are publicly available, they may have been included in the LLMs' pretraining data.

References

Lakshya A Agrawal, Shangyin Tan, Dilara Soylu, Noah Ziems, Rishi Khare, Krista Opsahl-Ong, Arnav Singhvi, Herumb Shandilya, Michael J Ryan, Meng Jiang, Christopher Potts, Koushik Sen, Alex Dimakis, Ion Stoica, Dan Klein, Matei Zaharia, and Omar Khattab. 2026. GEPA: Reflective prompt evolution can outperform reinforcement learning. In *The Fourteenth International Conference on Learning Representations*.

Xiaoyu Cai, Yao Fu, Hong Zhao, Weihao Jiang, and Shiliang Pu. 2022. Memory graph with message rehearsal for multi-turn dialogue generation. In *Proceedings of the 31st ACM International Conference on Information & Knowledge Management (CIKM)*, pages 108–117, Atlanta, GA, USA. Association for Computing Machinery. CC BY-SA 4.0.

Preetha Chatterjee, Kostadin Damevski, Nicholas A Kraft, and Lori Pollock. 2020. Software-related slack chats with disentangled conversations. In *Proceedings of the 17th international conference on mining software repositories*, pages 588–592.

Micha Elsner and Eugene Charniak. 2008. You talking to me? a corpus and algorithm for conversation disentanglement. In *Proceedings of the 46th Annual Meeting of the Association for Computational Linguistics (ACL)*, pages 834–842. Association for Computational Linguistics.

Micha Elsner and Eugene Charniak. 2010. Disentangling chat. In *Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics*, pages 117–126. Association for Computational Linguistics.

Qingyan Guo, Rui Wang, Junliang Guo, Bei Li, Kaitao Song, Xu Tan, Guoqing Liu, Jiang Bian, and Yujiu Yang. 2024. Connecting large language models with evolutionary algorithms yields powerful prompt optimizers. In *The Twelfth International Conference on Learning Representations*.

Lawrence Hubert and Phipps Arabie. 1985. Comparing partitions. *Journal of classification*, 2(1):193–218.

Jonathan K Kummerfeld, Sai R Gouravajhala, Joseph J Peper, Vignesh Athreya, Chulaka Gunasekara, Jatin Ganhotra, Siva Sankalp Patel, Lazaros C Polymenakos, and Walter Lasecki. 2019. A large-scale corpus for conversation disentanglement. In *Proceedings of the 57th annual meeting of the association for computational linguistics*, pages 3846–3856.

Brian Lester, Rami Al-Rfou, and Noah Constant. 2021. The power of scale for parameter-efficient prompt tuning. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pages 3045–3059.

Bobo Li, Hao Fei, Fei Li, Shengqiong Wu, Lizi Liao, Yinwei Wei, Tat-Seng Chua, and Donghong Ji. 2025. Revisiting conversation discourse for dialogue disentanglement. *ACM Transactions on Information Systems*, 43(1):1–34.

Tianda Li, Jia-Chen Gu, Xiaodan Zhu, Quan Liu, Zhen-Hua Ling, Zhiming Su, and Si Wei. 2020. Dialbert: A hierarchical pre-trained model for conversation disentanglement. *arXiv preprint arXiv:2004.03760*.

Xiang Lisa Li and Percy Liang. 2021. Prefix-tuning: Optimizing continuous prompts for generation. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 3881–3897.

Xinbei Ma, Zhuosheng Zhang, and Hai Zhao. 2021. Structural characterization for dialogue disentanglement. *arXiv preprint arXiv:2110.08018*.

Aaron F McDaid, Derek Greene, and Neil Hurley. 2011. Normalized mutual information to evaluate overlapping community finding algorithms. *arXiv preprint arXiv:1110.2515*.

Marina Meilă. 2003. Comparing clusterings by the variation of information. In *Learning Theory and Kernel Machines: 16th Annual Conference on Learning Theory and 7th Kernel Workshop, COLT/Kernel 2003, Washington, DC, USA, August 24-27, 2003. Proceedings*, pages 173–187. Springer.

OpenAI. 2025. Introducing GPT-5.2. Accessed: 2026-05-21.

Krista Opsahl-Ong, Michael J Ryan, Josh Purtell, David Broman, Christopher Potts, Matei Zaharia, and Omar Khattab. 2024. Optimizing instructions and demonstrations for multi-stage language model programs. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pages 9340–9366, Miami, Florida, USA. Association for Computational Linguistics.

Yawen Ouyang, Moxin Chen, Xinyu Dai, Yinggong Zhao, Shujian Huang, and Jiajun Chen. 2020. Dialogue state tracking with explicit slot connection modeling. In *Proceedings of the 58th annual meeting of the association for computational linguistics*, pages 34–40.

Melanie Sclar, Yejin Choi, Yulia Tsvetkov, and Alane Suhr. 2024. Quantifying language models' sensitivity to spurious features in prompt design or: How i learned to start worrying about prompt formatting. In *The Twelfth International Conference on Learning Representations*.

Dou Shen, Qiang Yang, Jian-Tao Sun, and Zheng Chen. 2006. Thread detection in dynamic text message streams. In *Proceedings of the 29th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 35–42.

Naoki Takada and Tatsunori Mori. 2026. Rethinking dialogue disentanglement for llms via dialogue-level assignment and subsequent context. In *In the Proceedings of The 10th Linguistic and Cognitive Approaches To Dialog Agents (LaCATODA 2026) Workshop at AAAI 2026 Conference*, pages 38–70, Singapore.

Zhi Rui Tam, Cheng-Kuang Wu, Yi-Lin Tsai, Chieh-Yen Lin, Hung-yi Lee, and Yun-Nung Chen. 2024. Let me speak freely? a study on the impact of format restrictions on large language model performance.

In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing: Industry Track*, pages 1218–1236, Miami, Florida, US. Association for Computational Linguistics.

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, Chujie Zheng, Dayiheng Liu, Fan Zhou, Fei Huang, Feng Hu, Hao Ge, Haoran Wei, Huan Lin, Jialong Tang, and 41 others. 2025. Qwen3 technical report. *Preprint*, arXiv:2505.09388.

Chengrun Yang, Xuezhi Wang, Yifeng Lu, Hanxiao Liu, Quoc V Le, Denny Zhou, and Xinyun Chen. 2024. Large language models as optimizers. In *The Twelfth International Conference on Learning Representations*.

Yongchao Zhou, Andrei Ioan Muresanu, Ziwen Han, Keiran Paster, Silviu Pitis, Harris Chan, and Jimmy Ba. 2023. Large language models are human-level prompt engineers. In *The Eleventh International Conference on Learning Representations*.

A Prompts Used in DD-GEPA

A.1 Prompt for Generating Utterance Representation Conversion Code

```
You are a coding assistant.
You will be given the proposed utterance-representation format. Your task is to write a program that consumes that input format
      and converts the chat data into the proposed representation.

Chat data format (before conversion):
- JSON format.
- It is a object under "CHAT_DATA" with two keys:
  - "candidate_utterances": array of utterances (up to 50 items)
  - "target_utterance": one utterance object
- Every utterance object has exactly these four keys:
  - "utterance_index"
  - "time"
  - "speaker"
  - "utterance_text"

Chat data example befoer conversion (illustrative only):
{
  "CHAT_DATA": {
    "candidate_utterances": [
      {
        "utterance_index": 11,
        "utterance_time": "2004-11-15T12:00",
        "speaker": "bob2",
        "utterance_text": "oga: please keep things in the channel"
      },
      {
        "utterance_index": 12,
        "utterance_time": "2004-11-15T12:14",
        "speaker": "lukus001",
        "utterance_text": "bob2: that link isn't of any use... it doesnt help me get the codecs ?:S"
      }
    ],
    "target_utterance": {
      "utterance_index": 13,
      "utterance_time": "2004-11-15T12:57",
      "speaker": "bob2",
      "utterance_text": "all I can suggest is reading the howto again and making sure you followed all the steps"
    }
  }
}

<<<BEGIN_PROPOSAL_TEXT>>>
{COMBINED_PROPOSAL_TEXT}
<<<END_PROPOSAL_TEXT>>>

Program contract:
- Do not hardcode the example chat content from this prompt or from the proposal text.
- The output must explicitly indicate which utterance is the target and must include natural-language wrapper text such as a "#
      chat data" header (or an equivalent header) consistent with the proposal intent.
- You may adapt wrapper wording as long as it stays consistent with the proposal intent.
- The program must define main().
- Read stdin JSON in the specified input format and write the transformed representation text (including wrapper text) to stdout
      .

Provide the program within ``` blocks.
```

A.2 Prompt for Generating Output Interpretation Code

```
You are a coding assistant.
You are given a prompt for models to use and a required output format. Your task is to write a program to convert the model's
      output into a required format.

You will receive a required output format. The model will produce an output for that task. Your task is to write a program that
      transforms the model's raw output into the required format.

Program contract:
- The program must have main().
- Assume the model output generated from the prompt shown below is structured. Even if the specific values change, design the
      program to recognize and use the same structure so it can reliably transform any output that follows that structure.
      However, the model's output formatting may vary; make the parser robust enough to handle minor deviations (e.g., missing
      commas or quotation marks) while still extracting the correct values.
- The program must take the model's output text as input and write the two-key JSON to stdout.

Provide the program within ``` blocks.

# Prompt for models
```text
{PRODUCTION_PROMPT_TEXT}
```
```

```
Reqired output format:
Identify the output-format from that prompt text and make the program to parse the model Output to two keys.
- "is_new_dialogue": boolean(true if the target is determined to be the start of a new dialogue, false if it is a continuation
      of an existing utterance).
- "utterance_id": If is_new_dialogue is true, set to null. If is_new_dialogue is false, set the ID of the selected candidate
      utterance.

Output example 1:
{
  "is_new_dialogue": false,
  "utterance_id": 5
}

Output example 2:
{
  "is_new_dialogue": true,
  "utterance_id": null
}
```

A.3 Prompt for Generating Failure Feedback

```
  Analyze the model's assignment and provide feedback.

Prompt given to the model (task instruction + chat data + output instruction):
'''
{decision_input}
'''

The model's output (include reasoning):
'''
{model_output}
'''
The assingment is incorrect. correct assignment is below:
{gold_assignment_text}

Explain why the assistant failed
Be specific, but don't make it too long.
```

A.4 Prompt for Updating Task Instruction

```
I provided an assistant with the following instructions to perform a task for me:

'''
<curr_instructions>
'''

The following are examples of different task inputs provided to the assistant along with the assistant's response for each of
      them, and some feedback on how the assistant's response could be better:
<inputs_outputs_feedback>

Your task is to write a new instruction for the assistant. Do not write or modify the other two parts. The new task definition
      must integrate well with the existing representation and output-format parts.

Read the inputs carefully and identify the input format and infer detailed task description about the task I wish to solve with
      the assistant.

Read all the assistant responses and the corresponding feedback. Identify all niche and domain specific factual information
      about the task and include it in the instruction, as a lot of it may not be available to the assistant in the future. The
      assistant may have utilized a generalizable strategy to solve the task, if so, include that in the instruction as well.

Write only the task definition (what dialogue disentanglement is, how to identify response relationships between utterances). Do
       not include any instructions about output format, output keys, schemas.

Provide the new instructions within '''text blocks.
```

A.5 Prompt for Updating Utterance Representation

```
I provided an assistant with the following utterance arrangement to perform a task for me:
'''
<curr_instructions>
'''

The following are examples of different task inputs provided to the assistant along with the assistant's response for each of
      them, and some feedback on how the assistant's response could be better:
<inputs_outputs_feedback>

Your task is to make new utterance arrangement for the assistant. Do not write any instructions that define or explain the task
      to identify response relationships between utterances.
```

```
Provide exactly one proposal (not multiple alternatives). Consider varied formatting changes, such as adding arbitrary tags,
      using JSON or CSV styles, or changing the number of sections/fields. Include short example that shows not only how
      utterances are listed but also any wrapper text such as headers (e.g., "# chat data"). Clearly indicate which utterance is
       the target. Make the representation unambiguous so a program can uniquely reconstruct the utterance sequence.

Read the inputs carefully and identify the input format and infer the task format I wish to solve with the assistant.

Read all the assistant responses and the corresponding feedback. Identify all niche and domain specific factual information
      about the task, as a lot of it may not be available to the assistant in the future.

Provide the new arrangement example within ‘‘‘text blocks.
```

A.6 Prompt for Updating Output Instruction

```
I provided an assistant with the following output instructions to perform a task for me:
‘‘‘
<curr_instructions>
‘‘‘

The following are examples of different task inputs provided to the assistant along with the assistant’s response for each of
      them, and some feedback on how the assistant’s response could be better:
<inputs_outputs_feedback>

Your task is to write new output instructions for the assistant.
Strict scope for this module: output format only.
Do not define or explain the dialogue-disentanglement task, do not describe how to find response relationships, and do not
      include any chat-representation instructions.
Only specify how the final answer must be formatted and provide a concrete output example.

Provide exactly one proposal (not multiple alternatives). Consider varied formatting changes, such as adding arbitrary tags,
      using JSON or CSV styles, or changing the number of sections/fields. Please improve the output format so that large
      language models can identify response relationships between utterances more accurately. Make the format unambiguous so a
      program can uniquely interpret them. Include the output-format instruction text and an example of the output it should
      produce.

Read the inputs carefully and identify the input format and infer the task format I wish to solve with the assistant.

Read all the assistant responses and the corresponding feedback. Identify all niche and domain specific factual information
      about the task, as a lot of it may not be available to the assistant in the future.

Provide the new instructions within ‘‘‘text blocks.
```

B Dialogue Disentanglement Prompt Settings

B.1 Seed 1

```
# Instruction
You are given a multi-user chat with each line labeled with an index number, timestamp, speaker’s name, and text message for an
      utterance. Your task is to identify the preceding utterance that the target responds to, or indicate that it starts a new
      dialogue.

# Chat
50 [2005-07-06T02:10] <Tomcat_> Dreco: There’s also a "Search for file" in the "Places" menu.
51 [2005-07-06T02:10] <system_message> === edulix [~edulix@136.Red-80-59-147.pooles.rima-tde.net]  has joined #ubuntu
52 [2005-07-06T02:11] <[noobuntu]> crimsun is not here..
53 [2005-07-06T02:12] <Dreco> I am trying to find out where wine dumps its fake.windows installation files ?
54 [2005-07-06T02:13] <Dreco> or where I can find fake.windows

...

96 [2005-07-06T02:23] <system_message> === minholi [~minholi@200.193.131.1]  has joined #ubuntu
97 [2005-07-06T02:23] <amnesia> no thanks, I can repair most of that
98 [2005-07-06T02:23] <system_message> === davro [~davro@cpc4-ches2-3-0-cust194.lutn.cable.ntl.com]  has joined #ubuntu
99 [2005-07-06T02:23] <holycow> cool, whatever
100 [2005-07-06T02:24] <amnesia> I’m just not sure I need to configure the interfaces for hal, since NM says I have no devices
      :)

The last utterance is the target utterance.

# Output
Output exactly one index number of the utterance that the target responds to. If the target starts a new dialogue, output the
      target utterance’s own index number.
Output example:
5
```

B.2 Seed 2

```
# Instruction
```

```
You are given a multi-user chat with each line labeled with an index number, timestamp, speaker's name, and text message for an
      utterance. Your task is to identify the preceding utterance that the target responds to, or indicate that it starts a new
      dialogue.

# Chat
50 [2005-07-06T02:10] <Tomcat_> Dreco: There's also a "Search for file" in the "Places" menu.
51 [2005-07-06T02:10] <system_message> === edulix [~edulix@136.Red-80-59-147.pooles.rima-tde.net]  has joined #ubuntu
52 [2005-07-06T02:11] <[noobuntu]> crimsun is not here..
53 [2005-07-06T02:12] <Dreco> I am trying to find out where wine dumps its fake.windows installation files ?
54 [2005-07-06T02:13] <Dreco> or where I can find fake.windows

...

96 [2005-07-06T02:23] <system_message> === minholi [~minholi@200.193.131.1]  has joined #ubuntu
97 [2005-07-06T02:23] <amnesia> no thanks, I can repair most of that
98 [2005-07-06T02:23] <system_message> === davro [~davro@cpc4-ches2-3-0-cust194.lutn.cable.ntl.com]  has joined #ubuntu
99 [2005-07-06T02:23] <holycow> cool, whatever
100 [2005-07-06T02:24] <amnesia> I'm just not sure I need to configure the interfaces for hal, since NM says I have no devices
      :)

The last utterance is the target utterance.

# Output
Output two keys.
- "is_new_dialogue": boolean(true if the target is determined to be the start of a new dialogue, false if it is a continuation
      of an existing utterance).
- "utterance_id": If is_new_dialogue is true, set to null. If is_new_dialogue is false, set the ID of the selected candidate
      utterance.

Output example 1:
"is_new_dialogue": false
"utterance_id": "5"
Output example 2:
"is_new_dialogue": true
"utterance_id": null
```

B.3 Baseline

```
# Instruction
You are given a multi-user chat with each line labeled with an index number, timestamp, speaker's name, and text message for an
      utterance. Your task is to identify the preceding utterance that the target responds to, or indicate that it starts a new
      dialogue.

# Chat data

## Previous utterances
{
  "sequential_utterances": [
    {
      "utterance_id": 50,
      "timestamp": "2005-07-06T02:10",
      "speaker": "Tomcat_",
      "message": "Dreco: There's also a \"Search for file\" in the \"Places\" menu."
    },
    {
      "utterance_id": 51,
      "timestamp": "2005-07-06T02:10",
      "speaker": "system_message",
      "message": "=== edulix [~edulix@136.Red-80-59-147.pooles.rima-tde.net]  has joined #ubuntu"
    },
    {
      "utterance_id": 52,
      "timestamp": "2005-07-06T02:11",
      "speaker": "[noobuntu] ",
      "message": "crimsun is not here.."
    },
    {
      "utterance_id": 53,
      "timestamp": "2005-07-06T02:12",
      "speaker": "Dreco",
      "message": "I am trying to find out where wine dumps its fake.windows installation files ?"
    },
    {
      "utterance_id": 54,
      "timestamp": "2005-07-06T02:13",
      "speaker": "Dreco",
      "message": "or where I can find fake.windows"
    },

    ...

    {
      "utterance_id": 96,
      "timestamp": "2005-07-06T02:23",
      "speaker": "system_message",
```

```
      "message": "=== minholi [~minholi@200.193.131.1]  has joined #ubuntu"
    },
    {
      "utterance_id": 97,
      "timestamp": "2005-07-06T02:23",
      "speaker": "amnesia",
      "message": "no thanks, I can repair most of that"
    },
    {
      "utterance_id": 98,
      "timestamp": "2005-07-06T02:23",
      "speaker": "system_message",
      "message": "=== davro [~davro@cpc4-ches2-3-0-cust194.lutn.cable.ntl.com]  has joined #ubuntu"
    },
    {
      "utterance_id": 99,
      "timestamp": "2005-07-06T02:23",
      "speaker": "holycow",
      "message": "cool, whatever"
    }
  ]
}

## Target utterance
{
  "utterance_id": 100,
  "timestamp": "2005-07-06T02:24",
  "speaker": "amnesia",
  "message": "I'm just not sure I need to configure the interfaces for hal, since NM says I have no devices :)"
}

# Output
Output two keys.
- "is_new_dialogue": boolean(true if the target is determined to be the start of a new dialogue, false if it is a continuation
      of an existing utterance).
- "utterance_id": If is_new_dialogue is true, set to null. If is_new_dialogue is false, set the ID of the selected candidate
      utterance.

Output example 1:
"is_new_dialogue": false,
"utterance_id": "5"
Output example 2:
"is_new_dialogue": true,
"utterance_id": null
```

B.4 Optimum

```
You are given a multi-user chat where each line is labeled with an index number, timestamp, speaker's name, and text message.
      Your task is to identify the specific preceding utterance that the target (last) utterance directly responds to.

The target may be responding to:
- An utterance by a different speaker
- One of its own previous utterances (continuing its own thought)
- Or it may be starting a new dialogue with no clear response relationship to any previous utterance

To determine the correct preceding utterance:
1. Analyze the conversational context and topic flow
2. Identify the most relevant preceding utterance that the target is directly addressing
3. Consider that the target might be continuing its own previous thought (so the response relationship might not be to the
      immediately preceding utterance)
4. If the target introduces a completely new topic with no connection to previous utterances, it has started a new dialogue

The target is always the last utterance in the chat. Do not assume the target responds to the immediately preceding utterance -
      the response relationship may be to an earlier utterance that sets up the context for the target's message.

# chat data
{
  "messages": [
    {
      "id": 50,
      "datetime": "2005-07-06T02:10",
      "author": "Tomcat_",
      "content": "Dreco: There's also a \"Search for file\" in the \"Places\" menu.",
      "is_target": false
    },
    {
      "id": 51,
      "datetime": "2005-07-06T02:10",
      "author": "system_message",
      "content": "=== edulix [~edulix@136.Red-80-59-147.pooles.rima-tde.net]  has joined #ubuntu",
      "is_target": false
    },
    {
      "id": 52,
      "datetime": "2005-07-06T02:11",
```

```
      "author": "[noobuntu]",
      "content": "crimsun is not here..",
      "is_target": false
    },
    {
      "id": 53,
      "datetime": "2005-07-06T02:12",
      "author": "Dreco",
      "content": "I am trying to find out where wine dumps its fake.windows installation files ?",
      "is_target": false
    },
    {
      "id": 54,
      "datetime": "2005-07-06T02:13",
      "author": "Dreco",
      "content": "or where I can find fake.windows",
      "is_target": false
    },

    ...

    {
      "id": 96,
      "datetime": "2005-07-06T02:23",
      "author": "system_message",
      "content": "=== minholi [~minholi@200.193.131.1]  has joined #ubuntu",
      "is_target": false
    },
    {
      "id": 97,
      "datetime": "2005-07-06T02:23",
      "author": "amnesia",
      "content": "no thanks, I can repair most of that",
      "is_target": false
    },
    {
      "id": 98,
      "datetime": "2005-07-06T02:23",
      "author": "system_message",
      "content": "=== davro [~davro@cpc4-ches2-3-0-cust194.lutn.cable.ntl.com]  has joined #ubuntu",
      "is_target": false
    },
    {
      "id": 99,
      "datetime": "2005-07-06T02:23",
      "author": "holycow",
      "content": "cool, whatever",
      "is_target": false
    },
    {
      "id": 100,
      "datetime": "2005-07-06T02:24",
      "author": "amnesia",
      "content": "I'm just not sure I need to configure the interfaces for hal, since NM says I have no devices :)",
      "is_target": true
    }
  ]
}

# Output Format
Output exactly one line containing a JSON object with the key "response_index" and a value that is either:
- An integer (the index of the preceding utterance the target responds to), or
- The string "new_dialogue" (if the target starts a new dialogue with no clear response relationship to any previous utterance)

Example 1: Target responds to utterance 5
{"response_index": 5}

Example 2: Target starts a new dialogue
{"response_index": "new_dialogue"}
```