CRyCHIc: Chat Reconstruction Using Tree in the Context of Human-AI Interaction

Anonymous ACL submission

Abstract

The conversation list function is widely built into most of the popular Large-Language-Model-based (LLM-based) chat applications. However, it can be hard for the users of these applications to find the chat history they want in the conversation list. One crucial reason for this problem is that sometimes the users tend to talk about multiple topics within one conversation. From this insight, we discussed the benefits of performing a chat tree construction on the chat history and filtering the history according to the tree before sending the user prompt with the history to the LLMs. We believe both LLM performance and user experience can be improved by doing so. A tree-constructing framework named CRyCHIc is then developed to construct the conversation tree efficiently. To test the performance of our framework, we also provide a test dataset called WildChatTree. Our model reaches 68.4% accuracy and 84.8% recall with only around 0.7B parameters on this dataset, reaching a performance similar to DeepSeek-V3. Our study offers direction for the future advancement of efficient chat tree construction. We will publicly release our code, dataset and models upon acceptance.

1 Introduction

The Large-Language-Model-based (LLM-based) chat applications are widely used nowadays (Achiam et al., 2023). Most of these chat applications have a conversation list function to help users manage their conversations with LLMs. In the ideal situation, the chat content in a conversation should focus on one specific question or demand. After answering the user's first prompt, the user is expected to put forward follow-up prompts that need to be responded to based on the first prompt and the first assistant response. However, in the actual user-assistant interaction scenario, users do not always start new conversations properly when discussing new topics with LLMs. This deviation from the initial design goal of the conversation function may bring several problems: 1) By default, when new user requests are sent to LLMs, all prior conversation history is included, which also carries the risk of introducing irrelevant information, leading the model to produce less accurate results (Wu et al., 2024; Yoran et al., 2024; Jiang et al., 2024). 2) Since users tend to discuss multiple questions within one conversation, it might be hard for users to find the fragments of chat history they want from the conversation list when needed. Consequently, the functionality of the conversation list fails to achieve its intended purpose.

Apart from the issue with the conversation list mentioned earlier, another potential factor that could negatively impact both user experience and the performance of LLMs is the user's tendency to ask follow-up questions based on their initial query. Though these subquestions concern the same topic and they need the same part of the chat history, answering these subquestions might not require the history of other subquestions. This phenomenon indicates the potential tree structure of the chat history, where the user's first question is the root of the tree, while the subquestions and their derivatives are the leaf nodes or the internal nodes of the tree. If a user prompt is not a subquestion or follow-up question of the first question, it can be recognised as the root of a new conversation. By organising the chat histories as trees, sending the chat histories filtered by their trees could remove the irrelevant information and thus improve LLM's performance, as shown in Figure 1. Besides, displaying the chat histories as trees could improve the user's reading experience since they can locate the subquestion they are concerned now more easily.

Observing these problems, developing a certain method that can automatically detect topic changes, start new conversations, and construct chat trees may effectively boost user experience, bring positive effects to the LLM performance

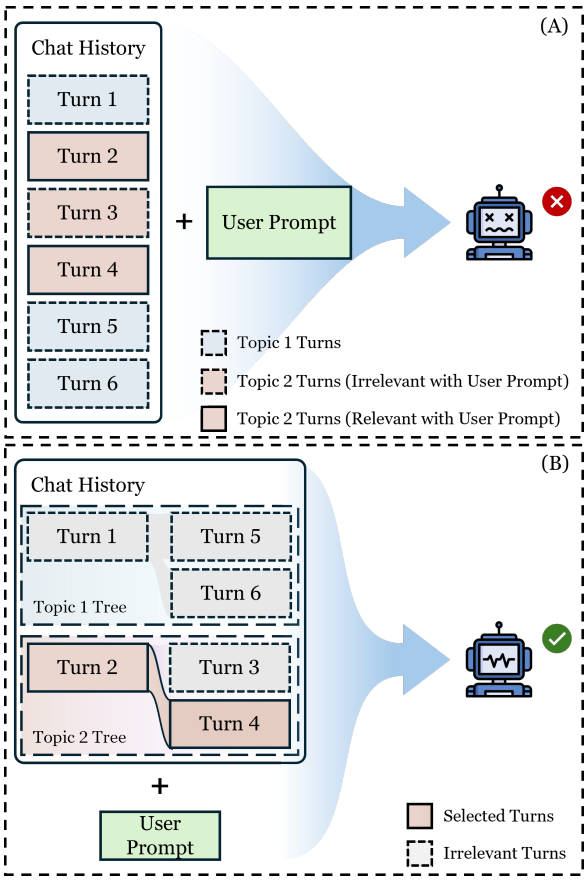

Figure 1: An example of using the tree to reconstruct and filter the chat history to improve LLM performance. (A) The original LLM input construction method. The current user prompt will be appended to the end of all the historical turns to construct the LLM input, no matter whether the historical turns are relevant to the user prompt or not. This may cause performance loss of the LLM due to irrelevant information in these historical turns. (B) LLM input with the tree-based chat history reconstruction. First, the chat history is organised as trees. Then, only the chat history tree branch relevant to the current user prompt will be sent to the LLM with the user prompt. This helps the LLM to only focus on the relevant chat history.

and even contribute to the long-context LLM practice. A naive method to achieve this goal is to simply apply an LLM to decide whether a new user prompt is a follow-up question of a certain chat turn in the chat history. However, this naive method can have performance issues. All the chat turns within a conversation need to be judged one by one to decide whether the new user prompt is their follow-up question. The total decision time can be very long, reaching an order of minutes, especially considering the response time of the LLM judges. This performance and the corresponding high computation cost are unacceptable in the production environment. To address the aforementioned problems, we proposed an efficient chat reconstructing framework called **C**hat **R**econstructor using **Tree** in the **C**ontext of **H**uman-AI **I**nter**a**ction (CRyCHIc). Inspired by Next Sentence Prediction (Shi and Demberg, 2019) and text embedding models, we designed and trained the core component in the CRyCHIc framework, the Follow-up Judge, using a high-efficiency two-stage architecture. We also introduced other components, such as the Implicit Follow-up Question Classifier (IFQC), to preprocess the user prompts after investigating possible types of user prompts. The overall pipeline of the CRyCHIc framework is shown in Figure 2.

To compare the performance of popular commercial LLMs with CRyCHIc, part of the mechanisms in the CRyCHIc framework are applied to enhance them. After testing CRyCHIc and these enhanced commercial LLMs on a human-labelled dataset, WildChatTree, we observed that this framework can effectively reduce the computation cost when reconstructing the chats into tree structures, while gaining comparable performance against most of the advanced commercial LLMs tested.

To summarise, our contributions are as follows:

(1) We analysed the failure of the conversation list function in the chat applications nowadays, and advocated that the chat history should be organised or reconstructed as a tree format to improve user experience and reduce the disruptive effect of irrelevant information in the chat history on the LLMs. This chat history reconstruction approach can also provide inspiration for building long-context LLMs.

(2) We proposed the CRyCHIc framework, an efficient framework that can construct chat trees with comparable accuracy and lower computation cost.

(3) We designed a novel two-stage architecture for the Follow-up Judge, and analysed the user prompts' types and inner structures, which may provide a reference for the design of future efficient chat reconstructors.

(4) We provided a human-labelled dataset, WildChatTree, containing 100 conversations and 567 turns to test the accuracy of chat tree construction.

2 Methodology

2.1 Task Definition

The task we would like to address in this work is reconstructing the chat history between the user

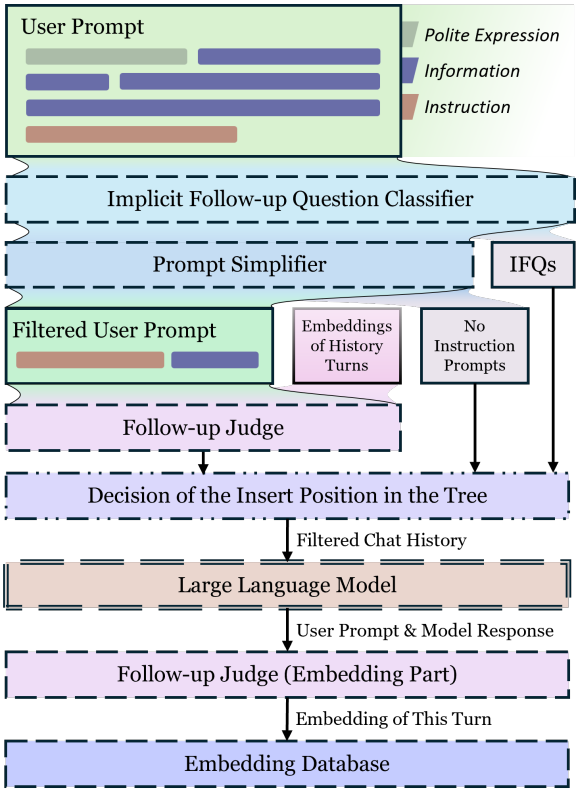

Figure 2: The pipeline of the CRyCHIc framework.

and the AI assistant into trees and filtering the chat history using the trees. Formally, the chat history H contains multiple conversation turns sorted in the creation time manner $\mathbf{t}_i = (p_i, r_i)$, where p_i refers to the user prompt in this turn and r_i refers to the model response in this turn. The task requires H to be reconstructed into trees of turns $\mathcal{T} = \{T_j, j \in \mathbb{N}^+\}$. Each tree T_j contains several turns and directed edges $(\mathbf{t}_n, \mathbf{t}_m)$. The edge between $\mathbf{t}_n$ and $\mathbf{t}_m$ exists only if the generation of r_n in $\mathbf{t}_n$ to respond p_n relies on the information or instruction from $\mathbf{t}_m$. If the edge exists, $\mathbf{t}_m$ can be called the parent of $\mathbf{t}_n$. The basic method to tackle this task is to traverse $\mathbf{t}_i$ in H to determine whether $\mathbf{t}_i$ is the parent of the new user prompt p_n. Aside from the basic definition of the task, several additional simplifications are applied in this work. If $\mathbf{t}_v$ relies on the information from its ancestor $\mathbf{t}_u$, where ancestor means there is a directed path from $\mathbf{t}_v$ to $\mathbf{t}_u$, it is not needed to establish an edge $(\mathbf{t}_v, \mathbf{t}_u)$. Furthermore, in the real scenario, one turn can have multiple parents. The CRyCHIc framework implemented in this work only considers the situation where one turn can only have one parent node to simplify the task. Correspondingly, the data in the WildChatTree test dataset also follows the same settings.

2.2 Framework Overview

As shown in Figure 2, the pipeline of the CRyCHIc framework has two stages. From the input of the user prompt to the decision of its insert position in the tree, the first stage (§2.3, §2.4) employs three modules to classify and simplify the user prompt and insert the user prompt into the right position in the tree. After sending the filtered chat history to the LLM to get the model's response, a few operations will be applied to the response to generate embeddings. The current turn, the reference of its parent and the embedding of the turn will be stored in the database next, preparing for future use (§2.5). Many prompt templates are used to instruct DeepSeek-V3 (DeepSeek-AI et al., 2025) to generate data for the modules, which are displayed in Appendix A.

2.3 User Prompt Preprocessing

2.3.1 User Prompt Analysis

To improve the efficiency and accuracy of the main component in the CRyCHIc framework, the Follow-up Judge (§2.3.4), we reviewed the user prompts when building the test dataset and proposed two modules to categorise and filter the user prompts before sending them to the Follow-up Judge.

First, some of the user prompts are simple follow-up questions like "Continue" and "Translate this". There are no explicit topic-related tokens in these prompts to show the topic or keywords they are discussing, seemingly allowing them to be follow-up questions to any turns in the chat history. In reality, however, users typically omit these topic-specific references deliberately, not because their requests can connect to any context, but because the subject they wish to discuss has already been clearly shown in the most recent turn. These user prompts are called implicit follow-up questions. The IFQC (§2.3.2) is developed to pick out these questions.

As shown in Figure 2, the sentences in the user prompt can be categorised into three categories, polite expression, information and instruction. The types of sentences within one user prompt can determine the type of the user prompt. To be specific, the instructions have the highest priority. If a user prompt contains an instruction, it will be assigned the instruction label. If it does not have instructions but has information, it will be assigned the information label. Finally, if it only has polite expressions

inside, it will be assigned the polite expression label. In the Prompt Simplifier (§2.3.3), only the prompts assigned with the instruction label will be sent to the Follow-up Judge, while the other two types of prompts will be assigned a position in the tree directly.

2.3.2 Implicit Follow-up Question Classifier

As mentioned above, the IFQC focus on classifying the user prompts into two categories, Explicit Follow-up Questions (EFQs) and Implicit Follow-up Questions (IFQs).

Module Architecture We believed that the IFQs are short sentences since long sentences will need topic-related words to build. Thus, we used a relatively small model, RoBERTa-base (Liu et al., 2019), as the base model, with a Multi-Layer Perceptron (MLP) as the classification head and 64 tokens as the max input length, to build the IFQC. In detail, we extract the last hidden state of the first token from RoBERTa's output as the embedding of the input sentence. An MLP classification head with 3 hidden layers is then responsible for classifying the embedding. Regarding the classification task of this module, the cross-entropy loss function is employed to fine-tune it.

Training Data Generation To train this model, we synthesised data using DeepSeek-V3. Three prompt templates are used to instruct the model to generate the data. They are positive prompts, aiming to generate the IFQs, negative prompts for EFQs and negative prompts for starting new topics. Finally, around 4.5k examples are used to train this module.

2.3.3 Prompt Simplifier

The Simplification Functionality Besides the aforementioned labelling function, Prompt Simplifier is also responsible for simplifying the EFQs filtered by the IFQC, for the Follow-up Judge. This function is also designed based on the idea of removing the irrelevant information. Given a simplified user prompt, the Follow-up Judge can focus more on the instruction sentences in the user prompt to improve the performance of the whole framework. The implementation of this functionality is simple. For those user prompts which have instruction sentences, all the instruction sentences and one information sentence (if it exists) will remain to construct the simplified user prompt.

Module Architecture First, the user prompt is segmented into sentences using the `en_core_web_md` from spaCy (Honnibal et al., 2020). A [SEP] token will then be concatenated to the end of these segmented sentences, and then all of them will be concatenated together. This operation can be expressed as:

$$p' = s_1 \oplus [SEP] \oplus s_2 \oplus [SEP] \oplus ... \oplus s_n \oplus [SEP], \quad where\ P = \{s_i, i = 1, 2, ...n\} \quad (1)$$

In this formula, p' represents the processed user prompt, P represents the set of sentences segmented from the original user prompt, and $\oplus$ represents the concatenation operation. Subsequently, p' will be input into the fine-tuned base model of the Prompt Simplifier, gte-base-en-v1.5 (Zhang et al., 2024), to get the last hidden states of every [SEP] token as the embeddings of each sentence. This concatenation operation can utilise the contextual information to provide help for the classification of every sentence. A similar MLP classification head as the one in Prompt Simplifier will classify every sentence using its embedding and assign it one of the three labels. The filtering mechanism mentioned before will then be applied to the set of sentences to build the filtered user prompt, and the user prompt will also be labelled accordingly.

Training Data Preparation We sampled 20k real English user prompts from the WildChat dataset (Zhao et al., 2024), a corpus of 1 million user-ChatGPT conversations, and labelled the sentences in them using DeepSeek-V3 similarly. We asked the LLM to score their certainty of the labelling result from 0-5, and only accept the results that have a score of 4 or 5. Finally, around 15k user prompts remained as the training data.

2.3.4 Follow-up Judge

Module Architecture Initially, we decided to employ the follow-up prediction approach, similar to the one used in the Next Sentence Prediction (NSP) task (Devlin et al., 2019; Sun et al., 2022). This approach requires using the model to traverse and process historical turns to check whether it is the parent of the current user prompt or not, resulting in high computational and time consumption. The details of this approach are discussed in §4.2.

To tackle the aforementioned problem, the Follow-up Judge employs two sub-modules, the embedding part and the classifier part. As the core component in the CRyCHIc framework aiming to

process domain-specific data, the embedding part is fine-tuned based on a larger base model, gte-large-en-v1.5 (Zhang et al., 2024). The classifier part is an MLP classifier with 5 hidden layers.

This module is allocated at two positions, after the Prompt Simplifier and in the LLM response postprocessing stage. When receiving the user prompt from Prompt Simplifier, the whole module will be applied to it, while only the embedding part will be allocated to generate embeddings for the model response, which will be introduced in §2.5. By employing this two-part architecture, the embeddings of the historical turns can be retrieved from the database, eliminating the need for recalculation and largely reducing the time consumption. Only the user prompt needs to be embedded when receiving it. Compared to the NSP-based approach mentioned above, although there is around 3% accuracy loss (94.8% $\rightarrow$ 91.8%) on the test split of this module when training it, we obtained great efficiency improvement.

In detail, the embedding part will be applied to the user prompt to get its embedding $\mathbf{e}^p$. Then, n embeddings of the most recent turns in the chat history $E_{recent} = \{\mathbf{e}_i^h, i \leq n \in \mathbb{N}^+\}$ will be retrieved from the embedding database. From the most recent history turn embedding to the least recent history turn embedding, each $\mathbf{e}_i^h$ will interact with $\mathbf{e}^p$ to generate the classifier part input, which is defined as follows:

$$\mathbf{e}_{interact} = \mathbf{e}_i^h \oplus \mathbf{e}^p \oplus (\mathbf{e}_i^h - \mathbf{e}^p), \qquad (2)$$

where $\mathbf{e}_i^h$ and $\mathbf{e}^p$ preserves the prompt and history turn information, while $\mathbf{e}_i^h - \mathbf{e}^p$ indicates the order of sequence between the two embeddings.

Consequently, the classifier part will assign a score for every historical turn, representing the probability of being the parent of the current user prompt. As mentioned in §2.1, only one parent will be assigned to each turn. Thus, only the historical turn with scores above the threshold and higher than other turns will be selected as the father of the user prompt. If no historical turn has a score higher than the threshold, the user prompt will be a root for a new conversation tree.

Training Data Preparation To train the Follow-up Judge with domain-specific data, we synthesise the data from several datasets. First, we sampled pairs of historical turns and user prompts from the WildChat dataset and filtered the data by employing DeepSeek-V3 as a judge to get 63k pairs. To help this module to familiar with different entities, we generate around 80k pairs based on the Wikipedia page categories. To train this module on the math and coding domain, we also transform the question-answer pairs in the APPS and MATH datasets (Hendrycks et al., 2021a; Hendrycks et al., 2021b) into chat histories. DeepSeek-V3 is then required to synthesise the user prompts based on the transformed histories. The APPS dataset provided around 9k pairs, and the MATH dataset provided around 14k pairs. In total, around 165k samples are used to train and test this module.

2.4 Deciding the Insert Position in the Tree

The user prompt preprocessing stage has divided the user prompts into the following categories:

1. Implicit Follow-up Questions;
2. Polite Expressions like "Hello", "Thanks";
3. Information provided for the former instructions in the chat history;
4. Explicit Follow-up Instructions with Parents;
5. Explicit Follow-up Instructions without Parents;

Before discussing how to decide the insert position in the chat history tree based on these categories, the highest priority is that no matter what category the user prompts belong to, if there is no turn in the chat history, they will become the root of a new tree. As to the user prompts in categories 1 and 3, they will be directly regarded as the children of the most recent turn in the chat history. For the user prompts in categories 2 and 4, they will become the root of a new tree. For the user prompt in category 3, the insert position is discussed in §2.3.4. After inserting the current user prompt into the correct place, all the ancestors (if they exist) will be retrieved to construct the filtered chat history and sent to the LLM with the user prompt.

2.5 Model Response Postprocessing

After acquiring the LLM-generated response based on the filtered chat history and user prompt, the embedding component of the Follow-up Judge processes both the user prompt and LLM response to generate embeddings for the current turn. The embedding generation process follows these steps: (1) Truncate the user prompt and the LLM response to fit within the max input length (512 tokens); (3) Concatenate the truncated user prompt with the truncated LLM response and append a [CLS] token to the end of this combined text; (4) Process this

sequence through the embedding component of the Follow-up Judge; and (5) Extract the last hidden state of the appended [CLS] token. The generated embedding will then be stored in the embedding database with the LLM response.

3 Experiments

In this section, we assess the CRyCHIc framework by exploring three key questions: (1) Can CRyCHIc have comparable performance with the leading commercial LLMs on the chat tree construction task? (2) How do individual sub-modules contribute to CRyCHIc's performance improvement? (3) Can CRyCHIc help to improve LLMs' performance on downstream tasks?

3.1 Performance on Chat Tree Construction Task

3.1.1 Dataset

To answer the first question, we developed a dataset called WildChatTree, containing 100 real human-LLM conversations and 567 turns. As its name indicates, WildChatTree is built on 100 English human-LLM conversations sampled and manually selected from the WildChat dataset (Zhao et al., 2024). Each original sample from WildChat is reviewed carefully by graduate-level human annotators. The main modification made on these samples is that every turn is assigned a parent turn if it needs to be generated based on the information from its parent. By asking the chat tree constructing models to assign parents turn by turn, we can calculate the accuracy, precision, recall and F1 score of these models.

3.1.2 Baseline

Model Size Analysis Considering the requirement on time efficiency of the chat tree construction task in the context of human-AI interaction, the size of the CRyCHIc framework needs to be analysed as follows before selecting the baseline models.

- IFQC: 126M parameters, using RoBERTa-base as the base model.
- Prompt Simplifier: 138M parameters, using gte-base-en-v1.5 as the base model.
- Follow-up Judge: 438M parameters, using gte-large-en-v1.5 as the base model of the embedding part.

To summarise, there are around 702M parameters in the CRyCHIc framework in total.

Baseline Model Selection To compare the performance of the CRyCHIc framework with the leading commercial LLMs, we chose several representative models. We chose DeepSeek-V3 (DeepSeek-AI et al., 2025) since most of our training data is synthesised by this model. We also chose GPT-4.1 (OpenAI, 2025), the newest flagship model from OpenAI. To compare the performance of CRyCHIc with smaller models, we chose GPT4o-mini (Hurst et al., 2024) and Qwen3-Turbo (Yang et al., 2025) since their computational requirements are relatively lower, though still higher than our model's. We observed Qwen3-Turbo level models have difficulty in instruction following, indicating that models smaller than these models cannot perform the task well. Therefore, we do not test models that have a similar size to the CRyCHIc framework. We also tested the situation if no chat tree reconstruction is applied to the chat history, which means that for every two turns, the former turn will become the parent of the latter turn.

Enhancement on Baseline Models To adapt these models to the chat tree construction task with higher efficiency, we asked them to label the user prompts with "polite expression", "information" or "instruction" before finding the parent turn of the user prompts. So the experiment results of these models in Table 1, whose name has a * mark, are generated with the aforementioned enhancement. The detailed method to apply the enhancement is demonstrated in the Appendix C.

3.1.3 Experiment Settings

For the CRyCHIc framework, several hyperparameters need to be determined for prediction. The historical turns retrieval number in the Follow-up Judge is 20, and the threshold for it is 0.4. The training of the modules in the CRyCHIc framework used PyTorch, and the training hyperparameters are reported in the Appendix B. For the baseline models, we use a historical context window of 10 turns. The threshold for determining follow-up situations is set at 4 on a 0-5 scale. When the LLM assigns a belief score of 4 or higher to its prediction, the current user prompt is classified as a child of the historical turn. Due to the resource limitation, only a single run with the temperature set to 1 is performed in this work.

3.1.4 Results

As shown in Table 1, the enhanced GPT-4.1 shows the best performance among all for metrics except

for recall, reaching 77.1% accuracy and 76.4% F1 score, while it is the most expensive and computationally-intensive model among the tested models. The enhanced DeepSeek-V3 delivered the second-best performance in accuracy and precision, achieving 69.7% accuracy and 67.7% precision. Meanwhile, CRyCHIc showed the highest recall, which is 84.8%, and the second-best performance in F1 score, which is 66.2%. As to the smaller models, such as GPT4o-mini and Qwen3-Turbo, they showed unacceptable performance in recall, one is 40.3% and the other is 35.0%. We also observed that the non-reconstructing strategy can only provide the lowest accuracy but the third-best performance in recall. To summarise, CRyCHIc can provide performance similar to the enhanced DeepSeek-V3, while largely reducing the computation consumption. We also tested the performance of the original DeepSeek-V3. The result indicates that the labelling strategy brought great performance improvement to these commercial LLMs, especially in recall.

Among the four metrics, the recall should be especially focused on. Though one of the goals of performing chat tree construction is to remove the irrelevant information, it is more crucial to preserve the necessary information to secure the basic performance. Recall can measure the model's ability to preserve the necessary information well. We can observe that CRyCHIc largely outperforms the enhanced DeepSeek-V3 in recall (+23.0%), demonstrating its ability to retain necessary information.

Model	Accuracy	Precision	Recall	F1 Score
GPT-4.1*	**0.771**	**0.695**	0.847	**0.764**
DeepSeek-V3*	0.697	0.677	0.618	0.646
GPT4o-mini*	0.586	0.521	0.403	0.454
Qwen3-Turbo*	0.578	0.585	0.350	0.438
DeepSeek-V3	0.661	0.672	0.476	0.558
Non-Reconstructuring	0.446	0.503	0.797	0.617
CRyCHIc	0.684	0.566	**0.848**	0.679

Table 1: Comparison of the performance between popular commercial LLMs and CRyCHIc on the WildChatTree dataset. Model names with * representing these models are enhanced by the labelling strategy similar to the one used in CRyCHIc. The results in bold represent the best performance, while the underlined results indicate the second-best performance.

3.2 Ablation Study

In this section, we tried to remove the modules in the CRyCHIc pipeline that do not harm the basic operation of CRyCHIc to prove the necessity of these modules.

The experiment result shown in Table 2 demonstrates that both the IFQC and the Prompt Simplifier have a positive contribution on all four metrics for the WildChatTree dataset. Between the two modules, IFQC contributes more to the performance of the CRyCHIc model, especially to the precision (+10.3%).

Model	Accuracy	Precision	Recall	F1 Score
CRyCHIc	**0.684**	**0.566**	**0.848**	**0.679**
w/o PS	0.635	0.481	0.714	0.575
w/o IFQC	0.616	0.463	0.714	0.562

Table 2: Comparison of the performance between the full CRyCHIc pipeline and the pipeline that disabled IFQC or Prompt Simplifier (PS).

3.3 Influence on Downstream Tasks

To test the influence of reconstructing the conversations into trees on the downstream tasks, we modified the MultiChallenge benchmark (Sirdeshmukh et al., 2025) to MixMC, as shown in Figure 3. In detail, for every two data points in MultiChallenge, the conversation histories inside them are mixed randomly, while the turns in each conversation keep the original relative position. This procedure generates two new data points, which only changed the conversation histories in the original datapoints, while the other keys, like the final user prompt and pass criteria, stay the same. To perform the test, we chose two models, CRyCHIc and GPT-4.1, to reconstruct the conversations in MixMC before sending the datapoints into the evaluation pipeline provided by MultiChallenge. We observed the following conclusions in the experiment result shown in Table 3.

- When processing with the two irrelevant topics within one conversation in the MixMC benchmark, the response model suffers performance loss on all 4 challenges.
- The current chat reconstruction models, even the best model, GPT-4.1, cannot reconstruct the trees well, causing an accuracy drop on three of the challenges. The possible reasons are: (1) Sometimes the final user prompt will be recognised as a totally new question with no relationship with any turn in the chat history; (2) The models will assign the most recent turn the parent of the user prompt classified as information, which can be a good

Benchmark	Chat Rec. Model	Response Model	Overall Score	Inference Mem.	Self-coherence	Inst. Retention	Reliable Ver. Editing
MixMC	CRyCHIc	GPT-4o	0.168	0.062	0.180	0.130	**0.300**
MixMC	GPT-4.1	GPT-4o	0.188	0.097	**0.220**	0.159	0.275
MixMC	-	GPT-4o	0.193	0.097	**0.220**	0.203	0.250
MultiChallenge	-	GPT-4o	**0.227**	**0.159**	**0.220**	**0.261**	0.268

Table 3: The influence of chat tree reconstruction using CRyCHIc and GPT-4.1 on the MixMC benchmark. The four challenges in the MultiChallenge benchmark are Inference Memory, Self-coherence, Instruction Intention, and Reliable Versioned Editing.

strategy processing conversations in the real scenario, but can result in errors when dealing with randomly mixed conversations.

- Both chat reconstruction models can improve the performance of the response model on reliable versioned editing.

4 Related Work

4.1 Benchmarks for Multi-turn LLMs

Assessing the ability of LLMs in the context of multi-turn conversation is gathering more concerns nowadays. Compared to the former single-turn benchmarks like AlpacaEval (Dubois et al., 2025), MT-Bench (Zheng et al., 2023) and MT-Bench++ (Sun et al., 2023) have expanded evaluations to include multi-turn scenarios across multiple subject areas. Meanwhile. These datasets cannot fulfil the significant need for more detailed and nuanced evaluation methods for multi-turn interactions between humans and AI systems. Thus, MT-Bench-101 (Bai et al., 2024) and MultiChallenge (Sirdeshmukh et al., 2025) have been developed to perform fine-grained LLM evaluation. However, they do not notice the multi-topic nature of the conversations conducted in LLM chat applications. Many users tend to talk about several different topics within one conversation context, highlighting the need for a more realistic multi-turn conversation benchmark.

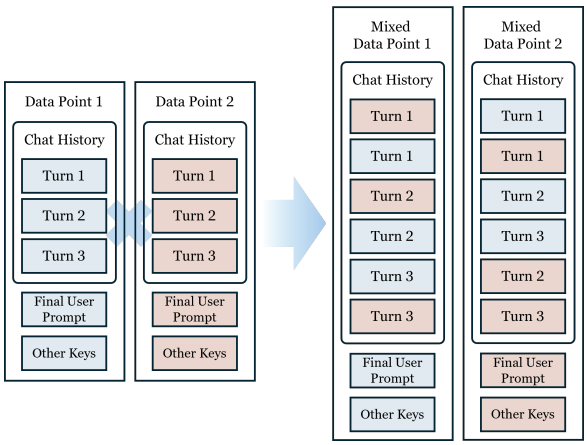

Figure 3: A demonstration of mixing original data points in the MultiChallenge benchmark to create the MixMC benchmark.

4.2 Next Sentence Prediction

The initial inspiration for the design of Follow-up Judge comes from the approach BERT used to tackle the Next Sentence Prediction (NSP) task (Devlin et al., 2019). To be specific, to solve the NSP task, the input of the language model should be constructed using the following equation (Sun et al., 2022):

$$\mathbf{x}_{input} = [\text{CLS}]\mathbf{x}_i^{(1)}[\text{SEP}]\mathbf{x}_i^{(2)}.[\text{EOS}], \quad (3)$$

where $\mathbf{x}_i^{(1)}$ represents the sentence 1 and $\mathbf{x}_i^{(2)}$ represents the sentence 2. The final hidden state of the [CLS] token will be extracted, and $\mathbf{s} = \mathbf{W}_{sem}(\tanh(\mathbf{W}h_{[\text{CLS}]}+\mathbf{b}))$ will be applied to generate a prediction.

In the chat tree construction task, $\mathbf{x}_i^{(1)}$ represents the historical turn and $\mathbf{x}_i^{(2)}$ represents the current user prompt.

5 Conclusion

This study proposes a promising task called chat tree (re)construction in the context of human-AI interaction. We believe that by performing this task to filter the chat history before sending the new user prompt to the LLM can effectively improve the performance of LLMs and reduce the computation consumption. Organising chat history as trees can also boost user experience. To reach the goal of constructing the chat tree efficiently, we designed and implemented the CRyCHIc framework, achieving comparable performance to DeepSeek-V3 with a much smaller model size. We also provided the WildChatTree dataset for testing the performance of models on the aforementioned task. We believe that our work paves the way for future explorations into the chat tree construction task.

Declaration

Declaration on Usage of Existing Artefacts

The existing artefacts used in this research and their licenses are listed below:

- PyTorch. License: Customised Apache 2.0
- SpaCy. License: MIT
- FAISS. License: MIT
- WildChat. License: ODC-BY
- MultiChallenge. License: Not specified
- MATH. License: MIT
- APPS. License: MIT
- Wikipedia Dumps. License: Creative Commons Attribution-Share-Alike 4.0
- GPT-4.1. License: Not specified
- DeepSeek-V3. License: MIT
- GPT4o-mini. License: Not specified
- Qwen3-Turbo. License: Apache 2.0
- RoBERTa-base. License: MIT
- gte-base-en-v1.5 and gte-large-en-v1.5. License: Apache 2.0

Though some artefacts do not specify a license, all of them can be used for research purposes.

Upon acceptance, the dataset, models and code of this paper will be released under the MIT license.

Declaration on AI Writing Assistance

In the writing process of this paper, AI is only used to give suggestions on word choices in this article, making it more formal.

Limitations

The experiment result in Table 1 demonstrates that the CRyCHIc framework can only reach 68.4% accuracy and 84.8% recall, which indicates that the CRyCHIc framework cannot be used in the production environment now. The experiment result on the MixMC benchmark also confirms this viewpoint. Possible reasons for this result include (1) the low quality of the synthesised data by DeepSeek-V3, (2) the model size limitation and context window limitation on the Follow-up Judge, and (3) ignorance of the history before the turn when predicting the follow-up relationship between the turn and the current user prompt. Besides, due to the resource limitation, the CRyCHIc framework is only trained on the English data. Future work could train the modules in CRyCHIc to adapt the framework to other languages and have a longer context window.

Although we have tested the influence of applying CRyCHIc and enhanced GPT-4.1 to reconstruct the conversation history in the MixMC benchmark, the MixMC is not built on real user-LLM chats, making the test results unreliable. A downstream benchmark that is generated from real human-AI interaction data is expected to be developed in the future.

Potential Risks

Performing chat tree construction to filter the chat history may deviate from the user's original thought when conducting conversations with AI assistants. Furthermore, the models used to perform chat tree construction may have potential bias, causing overemphasis on certain dialogue content.

References

Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, and 1 others. 2023. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774*.

Ge Bai, Jie Liu, Xingyuan Bu, Yancheng He, Jiaheng Liu, Zhanhui Zhou, Zhuoran Lin, Wenbo Su, Tiezheng Ge, Bo Zheng, and Wanli Ouyang. 2024. MT-bench-101: A fine-grained benchmark for evaluating large language models in multi-turn dialogues. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 7421–7454, Bangkok, Thailand. Association for Computational Linguistics.

DeepSeek-AI, Aixin Liu, Bei Feng, Bing Xue, Bingxuan Wang, Bochao Wu, Chengda Lu, Chenggang Zhao, Chengqi Deng, Chenyu Zhang, Chong Ruan, Damai Dai, Daya Guo, Dejian Yang, Deli Chen, Dongjie Ji, Erhang Li, Fangyun Lin, Fucong Dai, and 181 others. 2025. Deepseek-v3 technical report. *Preprint*, arXiv:2412.19437.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. Bert: Pre-training of deep bidirectional transformers for language understanding. *Preprint*, arXiv:1810.04805.

Matthijs Douze, Alexandr Guzhva, Chengqi Deng, Jeff Johnson, Gergely Szilvasy, Pierre-Emmanuel Mazaré, Maria Lomeli, Lucas Hosseini, and Hervé Jégou. 2025. The faiss library. *Preprint*, arXiv:2401.08281.

Yann Dubois, Balázs Galambosi, Percy Liang, and Tatsunori B. Hashimoto. 2025. Length-controlled alpacaeval: A simple way to debias automatic evaluators. *Preprint*, arXiv:2404.04475.

Dan Hendrycks, Steven Basart, Saurav Kadavath, Mantas Mazeika, Akul Arora, Ethan Guo, Collin Burns, Samir Puranik, Horace He, Dawn Song, and Jacob Steinhardt. 2021a. Measuring coding challenge competence with apps. *NeurIPS*.

Dan Hendrycks, Collin Burns, Saurav Kadavath, Akul Arora, Steven Basart, Eric Tang, Dawn Song, and Jacob Steinhardt. 2021b. Measuring mathematical problem solving with the math dataset. *NeurIPS*.

Matthew Honnibal, Ines Montani, Sofie Van Landeghem, Adriane Boyd, Henning Peters, Paul McCann, Maxim Jeanneau, Sneha Saxena, Karan Manning, Jim Gangi, Jim O'Regan, Pradeep Bhatia, Juniper Lourenço, Yiwei Shao, Duncan Rivers, Michael Samoylov, Daniel Rogers, Alan Wass, Michael Lewis, and John Howell. 2020. spacy: Industrial-strength natural language processing in python.

Aaron Hurst, Adam Lerer, Adam P Goucher, Adam Perelman, Aditya Ramesh, Aidan Clark, AJ Ostrow, Akila Welihinda, Alan Hayes, Alec Radford, and 1 others. 2024. Gpt-4o system card. *arXiv preprint arXiv:2410.21276*.

Ming Jiang, Tingting Huang, Biao Guo, Yao Lu, and Feng Zhang. 2024. Enhancing robustness in large language models: Prompting for mitigating the impact of irrelevant information. *Preprint*, arXiv:2408.10615.

Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. 2019. Roberta: A robustly optimized BERT pretraining approach. *CoRR*, abs/1907.11692.

OpenAI. 2025. Gpt-4-1. Accessed: May 15, 2025.

Jeff Rasley, Samyam Rajbhandari, Olatunji Ruwase, and Yuxiong He. 2020. Deepspeed: System optimizations enable training deep learning models with over 100 billion parameters. In *Proceedings of the 26th ACM SIGKDD international conference on knowledge discovery & data mining*, pages 3505–3506.

Wei Shi and Vera Demberg. 2019. Next sentence prediction helps implicit discourse relation classification within and across domains. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 5790–5796, Hong Kong, China. Association for Computational Linguistics.

Ved Sirdeshmukh, Kaustubh Deshpande, Johannes Mols, Lifeng Jin, Ed-Yeremai Cardona, Dean Lee, Jeremy Kritz, Willow Primack, Summer Yue, and Chen Xing. 2025. Multichallenge: A realistic multi-turn conversation evaluation benchmark challenging to frontier llms. *Preprint*, arXiv:2501.17399.

Yi Sun, Yu Zheng, Chao Hao, and Hangping Qiu. 2022. Nsp-bert: A prompt-based few-shot learner through an original pre-training task–next sentence prediction. *Preprint*, arXiv:2109.03564.

Yuchong Sun, Che Liu, Jinwen Huang, Ruihua Song, Fuzheng Zhang, Zhongyuan Wang, Di ZHANG, and Kun Gai. 2023. Parrot: Enhancing multi-turn chat models by learning to ask questions.

Wikimedia Foundation. 2022. Wikimedia database dump of the english wikipedia on may 01, 2022. Accessed: 2025-03-20.

Siye Wu, Jian Xie, Jiangjie Chen, Tinghui Zhu, Kai Zhang, and Yanghua Xiao. 2024. How easily do irrelevant inputs skew the responses of large language models? In *First Conference on Language Modeling*.

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.

Ori Yoran, Tomer Wolfson, Ori Ram, and Jonathan Berant. 2024. Making retrieval-augmented language models robust to irrelevant context. In *The Twelfth International Conference on Learning Representations*.

Dun Zhang, Jiacheng Li, Ziyang Zeng, and Fulong Wang. 2025. Jasper and stella: distillation of sota embedding models. *Preprint*, arXiv:2412.19048.

Xin Zhang, Yanzhao Zhang, Dingkun Long, Wen Xie, Ziqi Dai, Jialong Tang, Huan Lin, Baosong Yang, Pengjun Xie, Fei Huang, Meishan Zhang, Wenjie Li, and Min Zhang. 2024. mgte: Generalized long-context text representation and reranking models for multilingual text retrieval. *Preprint*, arXiv:2407.19669.

Wenting Zhao, Xiang Ren, Jack Hessel, Claire Cardie, Yejin Choi, and Yuntian Deng. 2024. Wildchat: 1m chatgpt interaction logs in the wild. *Preprint*, arXiv:2405.01470.

Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Siyuan Zhuang, Zhanghao Wu, Yonghao Zhuang, Zi Lin, Zhuohan Li, Dacheng Li, Eric Xing, and 1 others. 2023. Judging llm-as-a-judge with mt-bench and chatbot arena. *Advances in Neural Information Processing Systems*, 36:46595–46623.

A Details of Training Data Generation

A.1 IFQC Training Data Generation

As mentioned in §2.3.2, a set of prompts with several pre-determined variables is used to generate the training data for the IFQC. The prompt templates are shown below:

Prompt Template for Generating Positive Data

Generate 20 sentences. Follow the following instructions to generate the response.
- The sentences should be follow-up questions. It should contain a {ENDING_MARK}.
- The sentences should mimic the human user prompts when talking with an AI assistant.
- The sentences should not contain any nouns referring to a specific concept, like 'airplane' and 'gradients'.
- The sentences should not contain any verbs that are likely to appear in a certain domain.
- The sentences can contain pronouns and verbs.
- The length of the sentences should be in {LENGTH} words.
- All the generated sentences should be written in a list in a JSON format like ''sentences': [s1, s2 ...].'
- Do not repeat any sentence. Only give me json result.

Prompt Template for Generating Negative Data Which Still Contains Follow-up Questions

Generate 20 sentences. Follow the instructions below to generate the response. - The sentences should be follow-up questions, like asking for modification on the former explanation.
- The sentences should contain a {ENDING_MARK}.
- The sentences should mimic the human user prompts when talking with an AI assistant.
- The sentences should contain nouns referring to a specific concept relating to '{DOMAIN}'.
- The length of the sentences should be in {LENGTH} words.
- All the generated sentences should be written in a list in a JSON format like ''sentences': [s1, s2 ...].'
- Do not repeat any sentence. Only give me json result.

Prompt Template for Generating Negative Data Which Starts a New Topic

Generate 20 sentences. Follow the following instructions to generate the response.
- The sentences should contain a {ENDING_MARK}.
- The sentences should mimic the human user prompts when talking with an AI assistant.
- The sentences should contain nouns referring to a specific concept relating to '{DOMAIN}'.
- The length of the sentences should be in {LENGTH} words.
- All the generated sentences should be written in a list in a JSON format like ''sentences': [s1, s2 ...].'
- Do not repeat any sentence. Only give me json result."

The 'ENDING_MARK' is chosen from values ["question mark", "period"] and the 'LENGTH' is chosen from values ["3-5", "5-10", "10-20"]. For the prompt template aiming at generating negative data, the extra variable 'DOMAIN' is chosen from a set of domains like "Geometry" given by DeepSeek-V3.

A.2 Prompt Simplifier Training Data Generation

Before labelling the sentences in the user prompts using DeepSeek-V3, 'en_core_web_md' model from spaCy (Honnibal et al., 2020) is used to segment the user prompts. The prompt for labelling the sentences (Zhao et al., 2024) is shown below:

Prompt for Labelling Sentences from User Prompts

Analyze the given set of user sentences from a human-AI conversation and classify each sentence as one of the following types:

- Polite Expression: Greetings, thanks, or courteous phrases that don't contain specific requests or information
- Instruction: Sentences that direct the AI to perform a task, answer questions, or follow specific guidelines
- Information: Sentences that provide facts, context, or data to the AI.

For each sentence, determine its primary function in the context provided in the list.
Return the analysis in JSON format with the following structure:
json{{
"results": [
{{
"sentence": "Original sentence here",
"type": "One of: Polite Expression, Instruction, or Information",
"score": "0-5 score of confidence of your classification"
}}
]
}}
Consider the context of the conversation when making your classification.

Examples:
Polite Expression:
- "Thank you for your help with this problem."
- "Nice to meet you."

Instruction:
- "Please analyze this data and create a visualization."
- "Write a summary of the main points in this article."
- "Help me debug this code."
- "What is representation learning?"

Information:
"I'm working on a machine learning project for image recognition."
"The file contains sales data from January to December 2024."
"My computer is running Windows 11 with 16GB of RAM."
"10xp(xq+xr) dx (where p, q and r are positive constants)"
"I have experience with Python but I'm new to JavaScript."

The given set of user sentences:
{SENTENCE_LIST}

A.3 Follow-up Judge Training Data Generation

As mentioned in §2.3.4, the generation of training data for the Follow-up Judge is based on multiple data sources.

A.3.1 WildChat Data Processing

Before sampling data from the WildChat dataset, the original WildChat dataset is filtered by a series of operations to make sure only English conversations with over 2 turns remain for the following process. After sampling, a prompt will be applied to instruct DeepSeek-V3 to filter the sampled data, which is shown below:

Prompt for Labelling WildChat Data

Analyze the new prompt and determine whether it is a follow-up question to the provided chat history. Score the relationship on a scale of 0-5, where 0 means completely unrelated and 5 means a direct follow-up that information from the chat history is needed to answer the prompt.

Provide your analysis in JSON format with two fields:
1. "explanation": Your brief explanation on your score
2. "score": Your numerical score (0-5)

Chat History:
{CHAT_HISTORY}

New Prompt:
{USER_PROMPT}

Example response:
{{
"explanation": "Your explanation",
"score": 3
}}

A.3.2 Wikipedia Data Processing

To select 40k representative Wikipedia categories from 2450k categories in the Wikimedia database English dump on May 01, 2022 (Wikimedia Foundation, 2022), the original categories are first embedded by a text embedding model stella_en_400M_v5 (Zhang et al., 2025). Then, these embeddings are clustered by KMeans implemented by FAISS (Douze et al., 2025), using 100 iterations, and the categories whose embeddings are closest to the cluster centres are chosen to represent the clusters. Next, the selected 40k categories are regarded as the seed in the prompt, which will instruct DeepSeek-V3 to synthesise domain-specific pairs between chat history and new user prompt. The prompts are shown below:

Prompt for Generating Positive Pairs about Given Category

Task:
Generate a chat history and a follow-up prompt based on the theme "{CATEGORY}". The chat history should include a user message and an assistant response (limited to **200 words**), and the follow-up prompt cannot be answered without the context provided by the history.
Format:
```json
{{
"history": {{
"user": "User's message here",
"assistant": "Assistant's response here (max 200 words)"
}},
"prompt": "Prompt start a new topic can be answered without the history."
}}
```
Example:
```json
{}
```

Prompt for Generating Negative Pairs about Given Category (Same Theme)

Task:
Generate a chat history and a follow-up prompt based on the theme "{CATEGORY}". The chat history should include a user message and an assistant response (limited to **200 words**), and the prompt should start a new topic that has the same theme as the history and can be answered without the context provided by the history.
Format:
```json
{{
"history": {{
"user": "User's message here",
"assistant": "Assistant's response here (max 200 words)"
}},
"prompt": "Prompt start a new topic can be answered without the history."
}}
```
Example:
```json
{}
```

Prompt for Generating Negative Pairs about Given Category (Different Themes)

Task:
Generate a chat history based on the theme "{CATEGORY_1}". The chat history should include a user message and an assistant response (limited to **200 words**). Then generate a prompt based on another theme "{CATEGORY_2}" which starts a new topic and has no relevance with the history.
Format:
```json
{{
"history": {{
"user": "User's message here",
"assistant": "Assistant's response here (max 200 words)"
}},
"prompt": "Prompt start a new topic can be answered without the history."
}}
```
Example:
```json
{}
```

In the 80k generated pairs, there are 40k positive pairs and 40k negative pairs. The 40k negative pairs are also divided into two equal portions. The pairs in the first portion have chat history and user prompt with the same theme, and the pairs in the second portion have chat history and user prompt with different themes.

A.3.3 MATH and APPS Data Processing

To instruct DeepSeek-V3 to generate the following user prompts based on the chat history transformed from the MATH and APPS datasets (Hendrycks et al., 2021a; Hendrycks et al., 2021b), the following prompts are used:

Prompt for Generating Positive Pair

Generate a follow-up prompt based on the chat history below. The prompt can only be answered based on the chat history. For example, the prompt can be a question asking for an explanation of the solution in the chat history.

Chat History:
{HISTORY}

Format of your response:
```json
{{
"prompt": "Your follow-up prompt"
}}
```

Prompt for Generating Negative Pair

Generate a follow-up prompt based on the chat history below. The prompt should not have any relevance with the chat history and can concentrate on any domain.

Chat History:
{}

Format of your response:
```json
{{
"prompt": "Your follow-up prompt"
}} ```

## B Details of Training Settings

### B.1 Hyperparameters

The hyperparameters for training the IFQC are listed below:

- num_epochs: 5,
- learning_rate: 3e-5
- batch_size: 32,
- hidden_sizes: [768, 512, 256],
- dropout_rate: 0.2

The main hyperparameters for training the Prompt Simplifier are listed below. The batch size is the accumulated batch size since DeepSpeed (Rasley et al., 2020) is employed to accelerate the training.

- num_epochs: 20,
- learning_rate: 5e-5,
- batch_size (accumulated): 64,
- hidden_sizes: [768, 512, 256]

The main hyperparameters for training the Follow-up Judge are listed below. The batch size is the accumulated batch size since DeepSpeed is employed to accelerate the training.

- num_epochs: 20,
- learning_rate: 1e-5,
- batch_size (accumulated): 512,
- hidden_sizes: [1024, 512, 256, 64, 16]

### B.2 Hardware and Time Usage

The training of IFQC is done on a single NVIDIA GeForce RTX 4070 Ti 12GB graphics card, using 2 minutes. The training of the Prompt Classifier is done on 2 NVIDIA V100 32GB graphics cards, using around 2 hours. The training of the Follow-up Judge is also done on 2 NVIDIA V100 32GB graphics cards, using around 3 hours.

## C Details of Enhancement on Baseline Models

To enhance the baseline models with a labelling ability similar to the Prompt Simplifier, the following prompt is used to instruct the models to assign labels:
```

Prompt for Labelling

Analyze the given user prompt from a human-AI conversation and classify the prompt as one of the following types:

- Polite Expression: Greetings, thanks, or courteous phrases that don't contain specific requests or information
- Instruction: Sentences that direct the AI to perform a task, answer questions, or follow specific guidelines
- Information: Sentences that provide facts, context, or data to the AI

Return the analysis in JSON format with the following structure:
“‘json{{
"result":
{{
"sentence": "Original sentence here, if too long, truncate to 20 words. Regard the whole given user prompt as one sentence",
"type": "One of: Polite Expression, Instruction, or Information"
}}
}}“‘

Examples:
Polite Expression:
- "Thank you for your help with this problem."
- "Nice to meet you."

Instruction:
- "Please analyze this data and create a visualization."
- "Write a summary of the main points in this article."
- "Help me debug this code."
- "What is representation learning?"

Information:
"I'm working on a machine learning project for image recognition."
"The file contains sales data from January to December 2024."
"My computer is running Windows 11 with 16GB of RAM."
"10xp(xq+xr) dx (where p, q and r are positive constants)"
"I have experience with Python but I'm new to JavaScript."

The given user prompt:
{USER_PROMPT}

If the user prompt is labelled as an instruction, the prompt displayed below will be used for instructing the LLM to predict whether the current user prompt is a follow-up question of the given turn:

Prompt for Judging Whether the User Prompt is the Follow-up Question of the Given Historical Turn or Not

Analyze the new prompt and determine whether it is a follow-up question to the provided chat history. Score the relationship on a scale of 0-5, where 0 means completely unrelated and 5 means a direct follow-up that information from the chat history is needed to answer the prompt.

Provide your analysis in JSON format with two fields:
1. "explanation": Your brief explanation on your score
2. "score": Your numerical score (0-5)
The json response must have surrounding like “‘json.

Chat History:
{HISTORY}

New Prompt:
{USER_PROMPT}

Example response:
“‘json{{
"explanation": "Your explanation",
"score": 3
}}“‘