APPENDIX

This section provides supplementary materials, including detailed prompt designs, parameter settings for our method, the prompt of the baseline method, and the KG schemas generated by different methods.

Prompt Design for Our Method

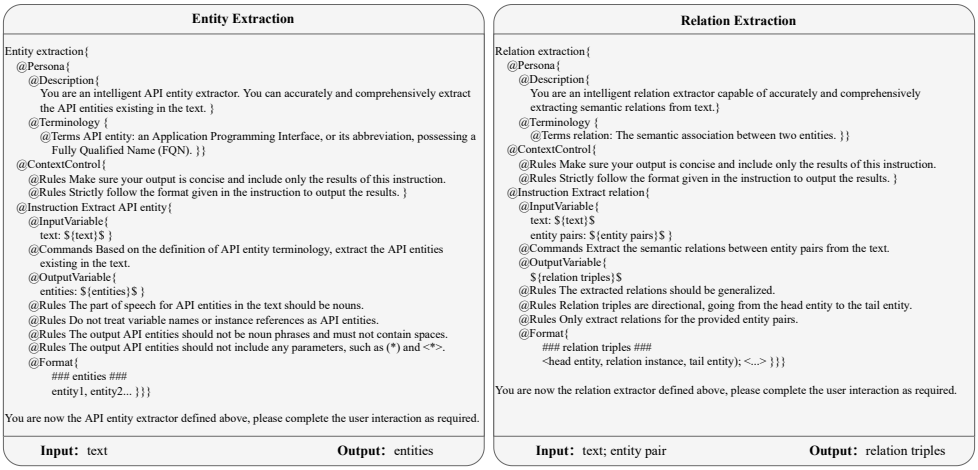

Fig. 1. Entity Extraction and Relation Extraction

All prompts in this paper use a structured design. Taking the left part of Fig. 1 as an example, it has three top-level parts: @Persona (which defines the identity and function of LLM), @ContextControl (which sets behavior constraints for LLM), and @Instruction (which provides operation instructions for LLM).

Among them, Persona contains two sub-parts:

- @Description: describes the task objective: (such as "You are an intelligent API entity extractor...");
- @Terminology: describes technical terms: (such as "Terms API entity...").

@ContextControl contains several @Rules that limit the behavior in the context, such as "Make sure your output is concise...";

@Instruction contains five sub-parts:

- @InputVariable: describes the input of prompt (such as "text" here);
- @Commands: clarifies the execution steps of the LLM, such as "Based on the definition of API entity terminology, extract the API entities...";
- @OutputVariable describes the input of prompt (such as 'entities" here);
- @Rules: emphasizes the notices when LLM executes the command, such as "The part of speech for API entities...", this rule can effectively avoid the common word ambiguity of API entities [1], for example, print" may be a verb or refer to java.io.printwriter.print();
- @Format: stipulates the format of the output result of LLM, such as "### entities ###...".

Through this structured prompt, LLM can generate API entities that are accurate and meet the format requirements. It should be noted that for complex tasks, the @Instruction part can be expanded with a sub-part @examples to guide the LLM to perform the task better. However, due to the tasks in this paper, such as entity extraction, etc., being relatively simple and the LLM can perform them well [2, 3], all prompts do not provide examples and belong to zero-shot.

Entity Type Labeling

```
Entity type labeling{
  @Persona{
    @Description{
      You are an intelligent entity type identifier that can label the types of entities in the text. }
@Terminology{
    @Term entity type: The entity type is the type of the entity's fully qualified name.}}
  @ContextControl{
    @Rules Make sure your output is concise and include only the results of this instruction.
    @Rules Strictly follow the format given in the instruction to output the results. }
  @Instruction Label entity types{
    @InputVariable{
      text: ${text}$
      entities: ${entities}$ }
    @Commands Based on the text, label the entity types of the given entities.
    @OutputVariable{
      ${entity types}$ }
    @Rules Ensure that all given entities are labeled with their entity types.
    @Rules Ensure, as much as possible, that each labeled entity type is specific.
    @Rules If an entity does not have a fully qualified name, label its entity type as "none."
    @Format{
        ### entity types ###
        entity1: type-1; entity2: type-2; ... }}}

You are now the entity type identifier defined above, please complete the user interaction as required.
```

Input: text; entities **Output:** entity types

Fig. 2. Entity Type Labeling

Entity Type Fusion

```
Entity type fusion{
  @Persona {
    @Description {
      You are an intelligent knowledge graph schema designer with the ability to reasonably fuse
      entity types. }}
  @ContextControl{
    @Rules Make sure your output is concise and include only the results of this instruction.
    @Rules Strictly follow the format given in the instruction to output the results. }
  @Instruction Fuse entity type{
    @InputVariable {
      entity types: ${entity types}$}
    @Commands Analyze existing entity types and fuse similar types into a new type.
    @Commands Generate an accurate and concise definition for each fused entity type.
    @OutputVariable {
      ${new entity type definitions}$
      ${new entity types and subtypes}$ }
    @Rules Entity types with similar characteristics or within the same conceptual hierarchy can be
      fused into a new entity type.
    @Rules Ensure not to over-fuse; do not fuse entity types that have clear distinctions.
    @Rules Do not overlook any given entity type.
    @Format{
        ### new entity type definitions ###
        new entity type: new entity type definition ...

        ### new entity types and their subtypes ###
        new entity type: [subtype1, subtype2, ...]}}}

You are now the entity type designer defined above, please complete the user interaction as required.
```

Input: fused entity types **Output:** fused entity types and their definitions

Relation Type Fusion

```
Relation type fusion{
  @Persona{
    @Description{
      You are an intelligent knowledge graph schema designer. You have the ability to reasonably fuse
relation types.}}
  @ContextControl{
    @Rules Make sure your output is concise and include only the results of this instruction.
    @Rules Strictly follow the format given in the instruction to output the results. }
  @Instruction Merge relation type{
    @InputVariable{
      relation types: ${relation types}$
    @Commands Analyze existing API relation types and fuse similar types into a new type.
    @Commands Generate an accurate and concise definition for each fused relation type.
    @OutputVariable {
      ${new relation type definitions}$
      ${new relation types and subtypes}$ }
    @Rules Relation types with similar semantics or same properties can be fused into a new type.
    @Rules Ensure that the naming and definitions of different relation types do not overlap, and that
      different relation types have clear distinctions.
    @Rules Fused new relation types should be specific. Avoid generating vague or abstract types.
    @Rules Define relations as generally as possible, using "one entity" and "the other entity" instead
      of specific entities.
    @Rules Ensure that all given relation types are considered during the fusion process.
    @Format{
        ### new relation type definitions ###
        new relation type: definition ...

        ### new relation types and their subtypes ###
        new relation type: [subtype1, subtype2, ...]}}}

You are now the relation type designer defined above, please complete the user interaction as required.
```

Input: fused relation types **Output:** fused relation types and their definitions

Fig. 3. Entity Type Fusion and Relation Type Fusion

Schema-guided Entity Extraction

```
Entity extractor{
  @Persona{
    @Description{
      You are an intelligent API entity extractor. You can accurately and comprehensively extract
      the entities existing in the text. }}
    @Terminology{
    @Terms API entity: an Application Programming Interface, or its abbreviation, possessing a
      Fully Qualified Name (FQN). }}
    @ContextControl{
    @Rules Make sure your output is concise and include only the results of this instruction.
    @Rules Strictly follow the format given in the instruction to output the results. }
  @Instruction Extract entity{
    @InputVariable{
        text: ${text}$
        entity types and their definitions: ${entity types and their definitions}$ }
    @Commands Extract API entities from the text based on the given entity types and definitions.
    @OutputVariable{
        ${entities with types}$ }
    @Rules You need to carefully analyze the text and ensure that no entities matching the
      predefined types are overlooked.
    @Rules Ensure that all entities matching the given type are extracted.
    @Rules The output API entities should not include any parameters, such as (*) and <*>.
    @Format{
        ### entities with types ###
        entity1: entity type; entity2: entity type; ... }}}

You are now the entity extractor defined above, please complete the user interaction as required.
```

Input: text **Output:** entity with types

Schema-guided Relation Extraction

```
Relation extractor{
  @Persona{
    @Description{
      You are an intelligent relation extractor that can extract relations from the text. }}
  @ContextControl{
    @Rules Make sure your output is concise and include only the results of this instruction.
    @Rules Strictly follow the format given in the instruction to output the results. }
  @Instruction Extract relation{
    @InputVariable{
        text: ${text}$
        entities pairs: ${entity pairs}$
        relation types and their definitions: ${relation types and their definitions}$ }
    @Commands Extract relation from the text for each entity pair based on the given relation types.
    @Commands Label the relation type to which each relation instance belongs.
    @OutputVariable{
        ${relation triples with relation types}$ }
    @Rules Only extract relation instances for the provided entity pairs
    @Rules Relation triples are directional, going from the head entity to the tail entity.
    @Rules Make sure that all entity pairs have been carefully analyzed.
    @Format{
        ### relation triples with types ###
        relation type: <head entity, relation instance, tail entity>... }}}

You are now the knowledge extractor defined above, please complete the user interaction as required.
```

Input: text **Output:** relation triples with types

Fig. 4. Schema-Guided Entity Extraction and Schema-Guided Relation Extraction

Parameter Setting

In this paper, we implement our method and baselines by calling GPT-4o [4]. It is the latest model of OpenAI, which has outstanding text understanding capabilities and can perform relatively complex inference tasks [5, 6]. When calling the LLM, some parameters usually need to be set, including temperature, max_tokens, n, frequency_penalty, and presence_penalty. Among them, temperature is used to control the randomness of the generated text. To ensure the stability of our method, we set it to 0 so that the LLM can generate more deterministic results. Max_tokens is used to specify the maximum length of the generated result. Since the result lengths output by different units are different, max_tokens has no fixed value. For example, the max_tokens of the entity extraction unit is set to 128; while for the entity type fusion unit and the relation type fusion unit, the max_token is set to 4096. The parameter n represents the number of generated results and is set to 1. In addition, frequency_penalty and presence_penalty are used to control the coherence of the generated text, and they are kept as the default values (i.e., 0).

Prompt of baseline Method

he prompt of GraphRAG [7] is shown in Fig. 5. To adapt it to API domain data, we restrict the scope of entities to APIs. Specifically, this prompt is used to extract API entities and their relations from a given text. Based on this prompt, the LLM first identifies and extracts all API entities. Then, based on the extracted entities, it constructs entity pairs, identifies the relations between them, and explains the connection between the source entity and the target entity. Additionally, to ensure a consistent format, it also presets delimiters, such as record_delimiter.

GraphRAG

-Goal-
Given a text and a list of API entity types, identify all API entities of those types from the text and all relationships among the identified entities. An API entity refers an Application Programming Interface, or its abbreviation, possessing a Fully Qualified Name (FQN).
-Steps-
1. Identify all entities. For each identified entity, extract the following information:
- entity_name: Name of the entity, capitalized
- entity_type: One of the following types: [{entity_types}]
Format each entity as ("entity"{tuple_delimiter}{tuple_delimiter})
2. From the entities identified in step 1, identify all pairs of (source_entity, target_entity) that are *clearly related* to each other.
For each pair of related entities, extract the following information:
- source_entity: name of the source entity, as identified in step 1
- target_entity: name of the target entity, as identified in step 1
- relationship_description: explanation as to why you think the source entity and the target entity are related to each other
Format each relationship as ("relationship"{tuple_delimiter}{tuple_delimiter}{tuple_delimiter})
3. Return output in English as a single list of all the entities and relationships identified in steps 1 and 2. Use **{record_delimiter}** as the list delimiter.
4. When finished, output {completion_delimiter}
-Output Format-
("entity"{tuple_delimiter}"xxx"{tuple_delimiter}"xxx")
{record_delimiter}
("entity"{tuple_delimiter}"xxx"{tuple_delimiter}"xxx")
{record_delimiter}
("relationship"{tuple_delimiter}"xxx"{tuple_delimiter}"xxx"
{tuple_delimiter}"xxxxxxxxxxxxxxxxxxxxxxxxx")
{record_delimiter}
("relationship"{tuple_delimiter}"xxx"{tuple_delimiter}"xxx"
{tuple_delimiter}"xxxxxxxxxxxxxxxxxxxxxxxxx")
{completion_delimiter}
######################
Entity_types: class; method; package
Text: {text}
Output:

Input: text **Output:** entities and relations

Fig. 5. The Prompt of GraphRAG

The content of the prompts for EDC [8] is shown in Fig. 6, 7, and 8. This series of prompts constitutes a complete information extraction process. Firstly, the LLM extracts API entities. Then, triples are extracted based on these entities and a detailed description is generated for each relation. Next, the LLM performs relation canonicalization to ensure their consistency. This process involves providing five candidate relations and their descriptions, and the LLM selects the most appropriate one to replace the relation in the given triple. When there are fewer than five types of relations, this step will be skipped.

Finally, the process moves into an iterative optimization phase. The LLM uses candidate entities and relations to further extract triples. Candidate entities include those extracted in the previous and current rounds, while candidate relations combine those previously extracted with relations identified as similar to the text using cosine similarity. Based on this information, the LLM refines the extraction of triples until no new triples are extracted.

Entity Extraction

Extract API entities from the given text. An API entity refers an Application Programming Interface, or its abbreviation, possessing a Fully Qualified Name (FQN)

In your answer, please strictly only include the entities and do not include any explanation or apologies.

Now extract entities from the following text. Text: {text}

Input: text **Output:** entities

Relation Extraction

Your task is to extract relational triples between the given API entities from the text. The triples should be in the form of [Entity1, Relation, Entity2].

Include only the triples in your response. Do not include any explanations or apologies.

Now please extract triples from the following text.
Text: {text}

Input: text and entities **Output:** triples

Fig. 6. Entity Extraction and Relation Extraction

Generate Relation Descriptions

You will be given a piece of text and a list of relational triples in the format of [Subject, Relation, Object] extracted from the text. For each relation present in the triples, your task is to write a description to express the meaning of the relation.

Now please extract relation descriptions given the following text and triples. Note that the description needs to be general and can be used to describe relations between other entities as well. Pay attention to the order of subject and object entities.

Text: {text}
Triples: {triples}

Input: text and triples **Output:** relation descriptions

Relation Canonicalization

Given the following text and a relational triple extracted from it:

Text: {text}
Triple: {triple}

The relation {relation} in the triplet is defined as {relation_definition}.
In this context, is there any relation appropriate to replace it? Please answer by providing only the letter of your choice.

Choices:
{choices}

Input: text and relations **Output:** new triples

Fig. 7. Relation Definition and Canonicalization

Refined Relation Extraction

Your task is to transform the given text into a semantic graph in the form of a list of triples.
The triples must be in the form of [Entity1, Relation, Entity2].
In your answer, please strictly only include the triples and do not include any explanation or apologies.

Now please extract triplets from the following text. Here are some potential relations and their descriptions you may look out for during extraction: {relations}
Note that this list may not be exhaustive, you may use other relations and not necessarily all relations in this list are present in the text.

Text: {text}
Candidate entities: {entities}

Input: text, candidate entities and relations

Output: triples

Fig. 8. Refined Relation Extraction

The KG Schema Generated by Each Method

The KG schema for each comparison method is shown in Table 1, and for each variant method in Table 2.

Table 1. Comparison of KG Schemas between Existing Methods

Method / Category	Manual-based		GraphRAG		EDC		Our	
	Number	Content	Number	Content	Number	Content	Number	Content
Entity Type	3	package, class, method	3	package, class, method	0	-	4	package, class, method, interface
Relation Type	11	efficiency comparison, function collaboration, contain, behavior difference, implement constraint, type conversion, logic constraint, function similarity, has method, function opposite, function replace	0	-	53	checks, precedes, test, inspects, is called before, created inside, provides, located in package, uses, operates on ...	13	preference, collaboration, containment, difference, implementation, conversion, dependency, equivalence, execution, limitation, creation, access, modification
Meta Triple	11	<class, efficiency comparison, class>, <class, function collaboration, class>, <package, contain, class>, <method, behavior difference, method>, <method, implement constraint, method>, <class, type conversion, class>, <method, logic constraint, method>, <method, function similarity, method>, <class, has method, method>, <method, function opposite, method>, <method, function replace, method>	0	-	0	-	30	<class, preference, class>, <class, collaboration, class>, <package, containment, class>, <method, difference, method>, <class, implementation, class>, <class, conversion, class>, <method, dependency, interface>, <class, equivalence, class>, <method, execution, method>, <method, limitation, method>, <method, creation, method>, <class, access, method>, <method, modification, class>, <class, difference, class>, <method, preference, method>, ...

Table 2. Comparison of KG Schemas between Variant Methods

Method / Category	Our		Our_w/o_KE		Our_w/o_KF		Our_w/o_FC	
	Number	Content	Number	Content	Number	Content	Number	Content
Entity Type	4	package, class, method, interface	3	package, class, method	4	package, class, method, interface	3	package, class, method, interface
Relation Type	13	preference, collaboration, containment, difference, implementation, conversion, dependency, equivalence, execution, limitation, creation, access, modification	11	efficiency comparison, function collaboration, contain, behavior difference, implement constraint, type conversion, logic constraint, function similarity, has method, function opposite, function replace	15	preference, collaboration, containment, difference, implementation, conversion, dependency, equivalence, execution, limitation, creation, access, modification, replacement, support	13	preference, collaboration, containment, difference, implementation, conversion, dependency, equivalence, execution, limitation, creation, access, modification
Meta Triple	30	<class, preference, class>, <class, collaboration, class>, <package, containment, class>, <method, difference, method>, <class, implementation, class>, <class, conversion, class>, <method, dependency, interface>, <class, equivalence, class>, <method, execution, method>, <method, limitation, method>, <method, creation, method>, <class, access, method>, <method, modification, class>, <class, difference, class>, <method, preference, method>, ...	11	<class, efficiency comparison, class>, <class, function collaboration, class>, <package, contain, class>, <method, behavior difference, method>, <method, implement constraint, method>, <class, type conversion, class>, <method, logic constraint, method>, <method, function similarity, method>, <class, has method, method>, <method, function opposite, method>, <method, function replace, method>	43	<class, preference, class>, <class, collaboration, class>, <package, containment, class>, <method, difference, method>, <class, implementation, class>, <class, conversion, class>, <method, dependency, interface>, <class, equivalence, class>, <method, execution, method>, <method, limitation, method>, <method, creation, method>, <class, access, method>, <method, modification, class>, <class, difference, class>, <method, preference, method>, <method, replacement, method>, <method, support, method>, ...	18	<class, preference, class>, <class, collaboration, class>, <package, containment, class>, <method, difference, method>, <class, implementation, class>, <class, conversion, class>, <method, dependency, interface>, <class, equivalence, class>, <method, execution, method>, <method, limitation, method>, <method, creation, method>, <class, access, method>, <method, modification, class>, ...

References

[1] Qing Huang, Yanbang Sun, Zhenchang Xing, Min Yu, Xiwei Xu, and Qinghua Lu. Api entity and relation joint extraction from text via dynamic prompt-tuned language model. *ACM Transactions on Software Engineering and Methodology*, 33(1):1–25, 2023.

[2] Qing Huang, Yanbang Sun, Zhenchang Xing, Yuanlong Cao, Jieshan Chen, Xiwei Xu, Huan Jin, and Jiaxing Lu. Let's discover more api relations: A large language model-based ai chain for unsupervised api relation inference. *ACM Transactions on Software Engineering and Methodology*, 2023.

[3] Qing Huang, Jiahui Zhu, Zhilong Li, Zhenchang Xing, Changjing Wang, and Xiwei Xu. Pcr-chain: Partial code reuse assisted by hierarchical chaining of prompts on frozen copilot. In *2023 IEEE/ACM 45th International Conference on Software Engineering: Companion Proceedings (ICSE-Companion)*, pages 1–5. IEEE, 2023.

[4] OpenAI. Openai gpt-4o model. https://platform.openai.com/docs/models/gpt-4o. Accessed: 2024.8.

[5] Anita Kirkovska Akash Sharma, Sidd Seethepalli. Analysis: Gpt-4o vs gpt-4 turbo. https://www.vellum.ai/blog/analysis-gpt-4o-vs-gpt-4-turbo. Accessed: 2024.8.

[6] OpenAI. Model evaluation results. https://github.com/openai/simple-evals. Accessed: 2024.8.

[7] Darren Edge, Ha Trinh, Newman Cheng, Joshua Bradley, Alex Chao, Apurva Mody, Steven Truitt, and Jonathan Larson. From local to global: A graph rag approach to query-focused summarization. *arXiv preprint arXiv:2404.16130*, 2024.

[8] Bowen Zhang and Harold Soh. Extract, define, canonicalize: An llm-based framework for knowledge graph construction. *arXiv preprint arXiv:2404.03868*, 2024.