APPENDIX

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

A. Prompt Design for Our Method

All prompts in this paper use a structured design [1]. Taking 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). Please refer to the code package [2] for more details.

Entity Extraction

```
Entity extraction{
  @Persona{
    @Description{
      You are an intelligent API entity extractor. You can accurately and comprehensively extract
      the API 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 API entity{
    @InputVariable{ text: ${text}$ }
    @Commands Based on the definition of API entity terminology, extract the API entities
              existing in the text.
    @OutputVariable{ entities: ${entities}$ }
    @Rules The part of speech for API entities in the text should be nouns.
    @Rules Do not treat variable names or instance references as API entities.
    @Rules The output API entities should not be noun phrases and must not contain spaces.
    @Rules The output API entities should not include any parameters, such as (*) and <*>.
     @Example{
      @Input{
        text: A thread that calls wait() on any object becomes inactive until another thread calls
              notify() on that object.}
      @Output{
        ### entities ###
        wait(), notify()}
      ......
}}}

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

Input: text **Output:** entities

Fig. 1: Prompt for Entity Extraction Unit.

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, e.g., "Ensure 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, for example, print" may be a verb or refer to java.io.printwriter.print();
- @Example: It is used to help understand the requirements of the task and clarify the output specifications.

Relation Extraction

```
Relation extraction{
  @Persona{
    @Description{
      You are an intelligent relation extractor capable of accurately and comprehensively
      extracting semantic relations from text.}
    @Terminology {
      @Terms relation: The semantic association between two entities. }}
  @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: ${text}$
      entity pairs: ${entity pairs}$ }
    @Commands Extract the semantic relations between entity pairs from the text.
    @OutputVariable{${relation triples}$ }
    @Rules The extracted relations should be generalized.
    @Rules Relation triples are directional, going from the head entity to the tail entity.
    @Rules Only extract relations for the provided entity pairs.
     @Example{
      @Input{
        text: If you need to read and write the date and time to a database, use the java.sql.Date
              and java.sql.Timestamp classes.
        entity pairs: (java.sql.Date, java.sql.Timestamp)}
      @Output{
        ### relation triples ###
        (java.sql.Date, works with, java.sql.Timestamp)}
      ......
}}}

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

Input: text; entity pairs **Output:** relation triples

Fig. 2: Prompt for Relation Extraction Unit.

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."
    @Example{
      @Input{
        text: Use Pattern.quote(".") to escape a period for splitting, and String.contains() to check if
              a string contains characters.
        entities: Pattern.quote(), String.contains()}
      @Output{
        ### entity types ###
        Pattern.quote(): static method; String.contains(): instance method }
      ......
}}}

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

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

Fig. 3: Prompt for Entity Type Labeling Unit.

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.
    @Example{
      @Input{
        entity types: concrete class, abstract class, utility class, abstract method, concrete method,
                instance method, static method}
      @Output{
        ### new entity types and subtypes ###
        class: [concrete class, abstract class, utility class]
        method: [abstract method, concrete method, instance method, static method]
        ### new entity type definitions ###
        class: A class defines the structure and behavior of objects, including fields, methods, and
            constructors, serving as a template for object instantiation.
        method: A method is a block of code that performs a specific task, defined within a class or
            interface, and can be invoked on objects or directly via the class (if static).}
      ......
}}}

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

Fig. 4: Prompt for Entity Type Fusion Unit.

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.
    @Example{
      @Input{
        relation types:
        [depends on, takes, relies on, converted by, converts, convert to, preferred over, better to use,
         recommended over]}
      @Output{
        ### new relation types and their definitions ###
        Dependency: One entity relies on another entity for its functionality or execution.
        Conversion: One entity is transformed into another entity or format.
        Preference: One entity is favored over another entity in terms of usage or efficiency.
        ### relation type mapping ###
        depends on: Dependency; takes: Dependency; relies on: Dependency; converted by:
        Conversion; converts: Conversion; convert to: Conversion; preferred over: Preference; better
        to use: Preference; recommended over: Preference}
      ......
}}}

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. 5: Prompt for Relation Type Fusion Unit.

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 <*>.
    @Example{
      @Input{
        text: A thread that calls wait() on any object becomes inactive until another thread calls
            notify() on that object.
        entity types and their definitions:
          class: A class defines the structure and behavior of objects, including fields, methods,
              and constructors, serving as a template for object instantiation.
          method: A method is a block of code that performs a specific task, defined within a class
              or interface, and can be invoked on objects or directly via the class (if static).
          interface: An interface defines a contract of methods that a class must implement. It can
              be functional or annotation-based.
          package: A package is a namespace that groups related classes and interfaces for better
              organization.}
      @Output{
        ### entities ###
        wait(): method; notify(): method}
      ......
}}}

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

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

Fig. 6: Prompt for Schema-guided Entity Extraction Unit.

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.
    @Example{
      @Input{
        text: If you need to read and write the date and time to a database, use the java.sql.Date and
            java.sql.Timestamp classes.
        entity pairs: (java.sql.Date, java.sql.Timestamp)
        relation types and their definitions:
          Dependency,One entity relies on another entity for its functionality or execution.
          Conversion,One entity is transformed into another entity or format.
          Preference,One entity is favored over another entity in terms of usage or efficiency.
          Implementation,One entity is realized or executed by another entity.
          Collaboration,One entity communicates or works with another entity.
          Containment,One entity includes or holds another entity within it.
          Modification,One entity alters or changes another entity.
          Creation,One entity generates or produces another entity.
          Difference,One entity is considered different to another entity.
          Replacement,One entity is substituted or exchanged for another entity.
          Execution,One entity initiates or carries out the operation of another entity.
          Access,One entity retrieves or obtains data from another entity.
          Support,One entity provides assistance or resources to another entity.
          Limitation,One entity imposes restrictions or constraints on another entity.
          Equivalence,One entity is considered equal or similar to another entity. }
      @Output{
        ### relation triples with relation types ###
        Collaboration: (java.sql.Date, works with, java.sql.Timestamp)}
      ......
}}}

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

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

Fig. 7: Prompt for Schema-guided Relation Extraction Unit.

B. Parameter Setting

In this paper, we implement our method and baselines by calling GPT-4o. It is the latest model of OpenAI, which has outstanding text understanding capabilities and can perform relatively complex inference tasks [3]. 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 (0).

C. The KG Schema Generated by Each Method

In this section, we will introduce the KG schema generated by each method. First, Table I presents the 13 relation types generated by our method, 9 of which overlap with the relation types in the existing MKC method [4]. For example, equivalence and function similarity both indicate that entities are similar or equal in functionality. In our method, the function_opposite relation summarized by MKC is merged into the "Difference" relation type. However, we also discover five unique relation types, including:

- Containment: It indicates that one entity contains another entity within it. For example, *SortedMap* contains *headMap()*.
- Modification: It means that One entity alters or modifies another entity. For instance, *remove()* can modify the elements in a *SortedSet*.
- Execution: It represents that one entity initiates or carries out the operation of another entity. For example, execute *lock()* to close the *Lock* instance.
- Access: It implies that one entity retrieves or acquires data from another entity. For example, *readInt()* reads data from a *DataInputStream*.
- Limitation: It signifies that one entity imposes constraints on another entity's behavior or functionality. For example, the output of *add()* is limited by the state of the *BlockingQueue*.

In fact, both "access" and "execution" could be merged into "modification". This is one of the limitations of our method.

Table II compares the differences in type information between existing methods and our method. EDC [5] is a schema-free method that focuses only on modeling relation types while neglecting entity types. In contrast, MKC defines 3 entity types (package, class, method) and 11 relation types, while our method defines 4 entity types (package, class, method, interface) and 13 relation types, generating a total of 34 type triples (24 of which are valid). This ensures the richness of the KG. Although the EDC method can refine relation types, there is still redundancy in the final relation types, which can be further optimized. For example, the relation types such as "checks", "precedes", and "test" have similar semantics and can be further merged. In contrast, our method can abstract low-dimensional relation types into high-dimensional ones, avoiding such semantic redundancy.

Table III shows the KG schemas designed by different variant methods. Due to $\text{Our}_{\text{w/oKE}}$ adopts the schema of MKC, which only contains 3 entity types and 11 relation types, resulting in 11 type triples. $\text{Our}_{\text{w/oKF}}$'s entity and relations types align with ours, but due to the lack of the KG filtering module, it includes 208 type triples, only 29 of which are valid, making the constructed KG unreliable. $\text{Our}_{\text{w/oFC}}$, although consistent with our entity and relation types, lacks a full-connectivity strategy, resulting in only 20 type triples (including 16 correct type triples), making it impossible to construct a comprehensive and rich KG.

Table IV shows the schemas generated by our method for different programming languages. The results indicate that, for Kotlin, it identifies 5 entity types (e.g., function) and 13 relation types, two of which are unique to Kotlin, such as extension and delegation. It generates a total of 34 type triples, 20 of which are valid. For Go, the method identifies 6 entity types (e.g., type) and 10 relation types, two of which are unique to Go, such as satisfaction and embedding. It generates a total of 55 type triples, 20 of which are correct. As for Java, the performance is consistent with what has been described previously. In summary, our method is still capable of constructing rich and reliable KGs even for less mainstream programming languages.

Table V demonstrates the comparison of the KG schemas designed based on different models. The results show that while all methods discover the same number of entity and relation types, the knowledge extraction differences lead to discrepancies. $\text{Our}_{\text{Llama}}$ retains 30 type triples, but only 20 of them are correct. $\text{Our}_{\text{Claude}}$ retains 32 type triples, with 23 being correct. As a result, the KGs constructed by these methods are slightly less rich and reliable compared to the KG constructed by our method.

REFERENCES

[1] Zhenchang Xing, Yang Liu, Zhuo Cheng, Qing Huang, Dehai Zhao, Daniel SUN, and Chenhua Liu. When prompt engineering meets software engineering: CNL-p as natural and robust "APIs" for human-AI interaction. In *The Thirteenth International Conference on Learning Representations*, 2025.

[2] Replication package. https://github.com/ybsun0215/Explore-Construct-Filter.

[3] 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.

[4] Qing Huang, Zhiqiang Yuan, Zhenchang Xing, Zhengkang Zuo, Changjing Wang, and Xin Xia. 1+ 1¿ 2: Programming know-what and know-how knowledge fusion, semantic enrichment and coherent application. *IEEE Transactions on Services Computing*, 16(3):1540–1554, 2022.

[5] Bowen Zhang and Harold Soh. Extract, define, canonicalize: An llm-based framework for knowledge graph construction. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2024.

TABLE I: The Details of the Relation Types in Our KG Schema.

Type (type in MKC [4])	Definition	Example
Equivalence (function similarity)	One entity is equal or very similar to another entity in functionality.	The offerLast() method adds an element to the end of the Deque, just like offer().
Difference (behavior difference)	One entity is different from another, typically in behavior or characteristics.	The add() and offer() methods behave differently when the queue is full.
Replacement (function replace)	One entity can substitute another in certain contexts without changing the expected result.	In many cases, you can replace the File class with the Path interface.
Preference (efficiency comparison)	One entity is favored over another due to efficiency or ease of use in a specific context.	BufferedInputStream is faster than reading single bytes from an InputStream...
Dependency (logic constraint)	One entity depends on another for its functionality or operation.	Collections.sort() relies on Arrays.asList() to sort array elements when more complex sorting is required.
Implementation (implement constraint)	One entity provides a concrete realization or behavior for another entity.	The PoolThreadRunnable class implements the Runnable interface, allowing it to be executed by a thread.
Collaboration (function collaboration)	One entity communicates or works with another entity to complete a specific task.	To set a date on a PreparedStatement or get a date from a ResultSet, you interact with java.sql.Date.
Conversion (type conversion)	One entity is transformed into another entity or format.	You can convert a Set to a List by passing the Set to the addAll() method of a new List.
Containment (contain/has method)	One entity contains another entity within it.	The headMap() method of SortedMap returns a new map containing the first elements of the original map.
Modification	One entity alters or modifies another entity.	To remove an element from a SortedSet, you call its remove() method, passing the element to be removed.
Execution	One entity initiates or carries out the operation of another entity.	To lock the Lock instance, you must call its lock() method.
Access	One entity retrieves or acquires data from another entity.	You can read data from a DataInputStream using its readInt() method.
Limitation	One entity imposes constraints on another entity's behavior or functionality.	If the BlockingQueue does not have space for a new element, the add() method throws an IllegalStateException.

TABLE II: Comparison of Type Information between the Existing Method and Our Method

Category	MKC		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, behavior difference, implement constraint, type conversion, logic constraint, function similarity, 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, replacement, difference, implementation, conversion, dependency, equivalence, execution, limitation, containment, access, modification
Type 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), (package, contain, method), (class, has method, method)	9	(class, null, class), (class, null, method), (class, null, package), (method, null, method), (method, null, class), (method, null, package), (package, null, package), (package, null, class), (package, null, method)	53	(null, check, null), (null, precedes, null), (null, test, null), (null, inspects, null), (null, is called before, null), (null, created inside, null), (null, provides, null), (null, located in package, null), (null, uses, null), ...	34	(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, replacement, method), (class, access, method), (method, modification, class), (class, difference, class), (method, preference, method), ...

TABLE III: Comparison of KG Schemas between Variant Methods

Category	$\text{Our}_{\text{w/oKE}}$		$\text{Our}_{\text{w/oKF}}$		$\text{Our}_{\text{w/oFC}}$		Our	
	Number	Content	Number	Content	Number	Content	Number	Content
Entity Type	3	package, class, method	4	package, class, method, interface	4	package, class, method, interface	4	package, class, method, interface
Relation Type	11	efficiency comparison, function collaboration, behavior difference, implement constraint, type conversion, logic constraint, function similarity, function replace, function opposite, contain, has method	13	preference, collaboration, replacement, difference, implementation, conversion, dependency, equivalence, execution, limitation, containment, access, modification	13	preference, collaboration, containment, difference, implementation, conversion, dependency, equivalence, execution, limitation, replacement, access, modification	13	preference, collaboration, containment, difference, implementation, conversion, dependency, equivalence, execution, limitation, replacement, modification, access
Type 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), (method, function replace, method), (package, contain, class), (class, has method, method)	208	(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, replacement, method), (class, access, method), (method, modification, class), (class, difference, class), (method, preference, method), ...	20	(class, preference, class), (method, collaboration, method), (package, containment, method), (method, difference, method), (class, implementation, class), (class, conversion, class), (method, dependency, method), (method, equivalence, method), (method, execution, method), (method, limitation, method), (method, replacement, method), (method, modification, class), ...	34	(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, replacement, method), (class, access, method), (method, modification, class), (class, difference, class), (method, preference, method), ...

TABLE IV: Comparison of KG Schemas across Different Languages

Category	Kotlin		Go		Java	
	Number	Content	Number	Content	Number	Content
Entity Type	5	package, class, function, interface, property	6	type, const, var, package, method, func	4	package, class, method, interface
Relation Type	13	preference, collaboration, containment, difference, implementation, conversion, dependency, equivalence, extension limitation, replacement, modification, delegation	10	collaboration, containment, difference, conversion, dependency, equivalence, limitation, replacement, satisfaction, embedding	13	preference, collaboration, containment, difference, implementation, conversion, dependency, equivalence, execution, limitation, replacement, modification, access
Type Triple	34	(function, dependency, class), (class, collaboration, class), (package, containment, class), (function, difference, function), (class, conversion, class), (method, dependency, interface), (class, equivalence, class), (function, extension, function), (method, limitation, method), (function, replacement, function), (method, modification, class), (class, delegation, class), (method, preference, method), ...	55	(method, collaboration, method), (type, containment, var), (method, difference, method), (type, satisfaction, type), (type, embedding, type), (type, conversion, type), (method, dependency, type), (method, equivalence, method), (func, limitation, var), (type, replacement, type), (type, difference, type), ...	34	(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, replacement, method), (class, access, method), (method, modification, class), (class, difference, class), (method, preference, method), ...

TABLE V: Comparison of KG Schemas across Different Models

Category	$\text{Our}_{\text{Llama}}$		$\text{Our}_{\text{Claude}}$		Our	
	Number	Content	Number	Content	Number	Content
Entity Type	4	package, class, method, interface	4	package, class, method, interface	4	package, class, method, interface
Relation Type	13	preference, collaboration, containment, difference, implementation, conversion, dependency, equivalence, execution, limitation, replacement, access, modification	13	preference, collaboration, containment, difference, implementation, conversion, dependency, equivalence, execution, limitation, replacement, access, modification	13	preference, collaboration, containment, difference, implementation, conversion, dependency, equivalence, execution, limitation, replacement, modification, access
Type 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, replacement, method), (class, access, method), (method, modification, class), (class, difference, class), (method, preference, method), ...	32	(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, replacement, method), (class, access, method), (method, modification, class), (class, difference, class), (method, preference, method), ...	34	(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, replacement, method), (class, access, method), (method, modification, class), (class, difference, class), (method, preference, method), ...