蚂蚁开源
ANT OPEN SOURCE

OpenSPG-KAG

KAG: Boosting LLMs in Professional Domains via Knowledge Augmented Generation

Speaker： zhengke.gzk@antgroup.com

Department: NextEvo-Language and Machine Intelligence-Knowledge Engine

蚂蚁开源
ANT OPEN SOURCE

Contents

蚂蚁开源 ANT OPEN SOURCE

Inherent flaws of the RAG + LLM Paradigm

LLM apps are typically equipped with private knowledge bases to address:

- The difficulty of using privacy data as pre-training corpus for open-source & commercial LLM
- The high requirements for personnel capabilities and resource allocation in LLM SFT
- The lengthy time cost in LLM SFT, making it challenging to stay synchronized with corpus updates

Are Text and Vector Indexes Relied by RAG effective KB？❌

Uniformly Management of Domain-specific Knowledge

Structured data, unstructured data, and domain expert rules cannot be uniformly stored and retrieved using text and vector indexes.

Lacking symbolic reasoning expression & execution

Recall strategy based on semantic similarity cannot handle complex reasoning, quantitative analysis. For example, how many males in "Dream of the Red Chamber"?

Neglect of the quality of knowledge base data.

homonyms with different meanings (e.g., Apple (Company) and Apple (Fruit)), and synonyms with different names (e.g., President Washington and The American George) need to be solved.

Knowledge graphs, enhanced with semantics, logic, and symbols, can provide better support for the LLM applications in professional domain

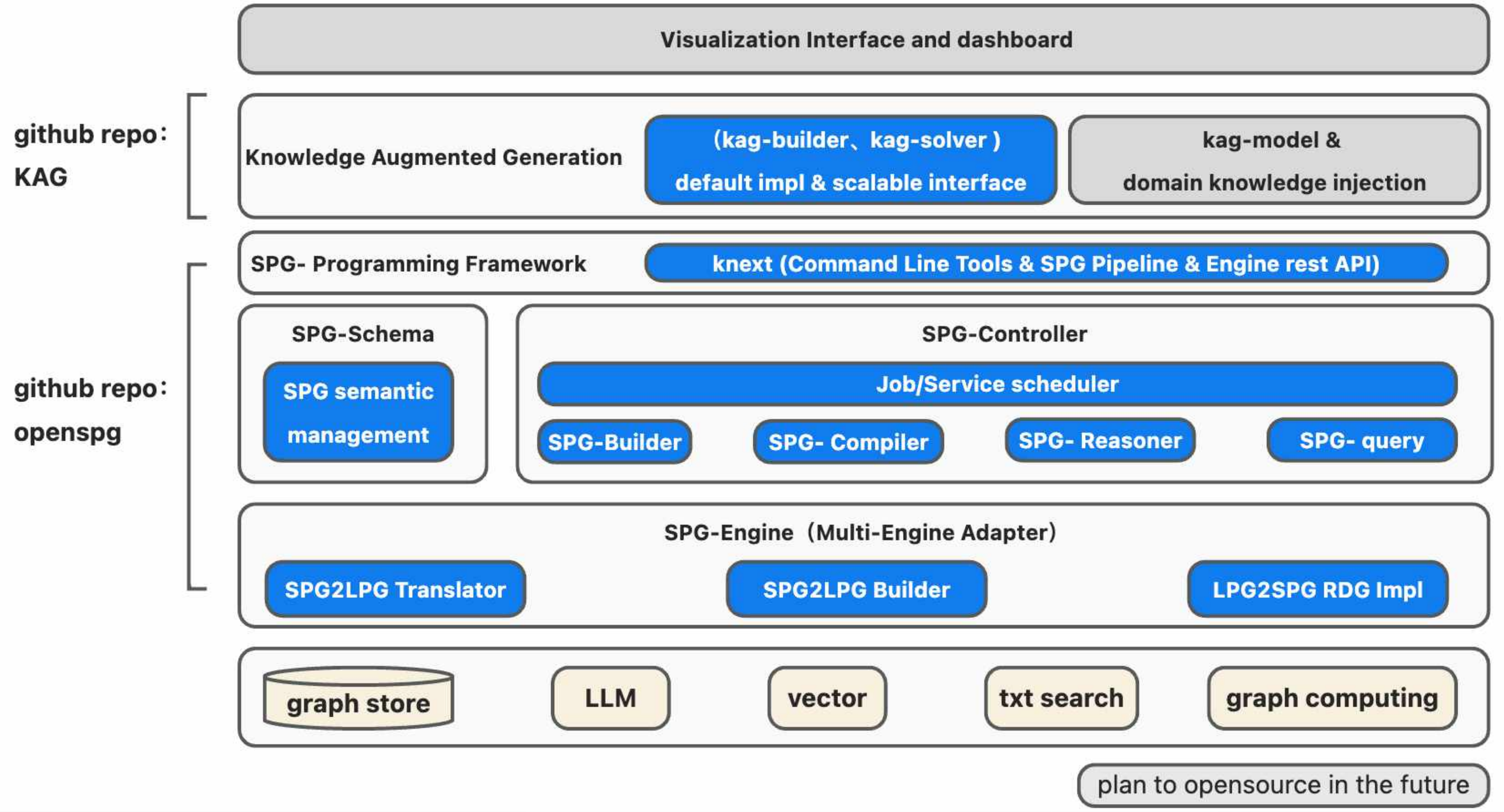
Knowledge Graph/Graph + LLM Typical Technical Approaches

Framework	applicable scenarios	Benchmarks (hotpotqa-1000 docs)	Characteristics
GraphRAG(MS)	QFS tasks (evaluation: Comprehensiveness, Diversity, Empowerment)	em: 0 f1: 0.053	• Through hierarchical clustering, progressively generate paragraph summaries for cross-document QFS tasks. • Lack of capability for logical symbolic reasoning.
HippoRAG	Factual QA tasks (evaluation: em, f1)	em: 0.457 f1 : 0.592	• Construction of the knowledge graph is based on RDF extraction and entities embedding linking. Chunk retrieved by combination of DPR + PPR during QA phase.
LightRAG	QFS tasks (evaluation: Comprehensiveness, Diversity, Empowerment)	em: 0 f1 : 0.034 Time cost: 4811 s Tokens: 1,772.3 k	• Extract RDF quintuples (with types) for construction. Achieve chunk retrieval by combination of ner and concept the ners.
KAG (V0.5)	Factual QA tasks (evaluation: em, f1)	Em: 0.625 f1 : 0.762 Time cost: 4232 s Tokens: 2,276 k	• KAG built on spg extraction, semantic alignment, and text & graph mutual indexing. • Factual-QA tasks completed through hybrid reasoning guided by logical symbols. • QFS tasks and dialogue QA tasks are yet to be open-sourced.

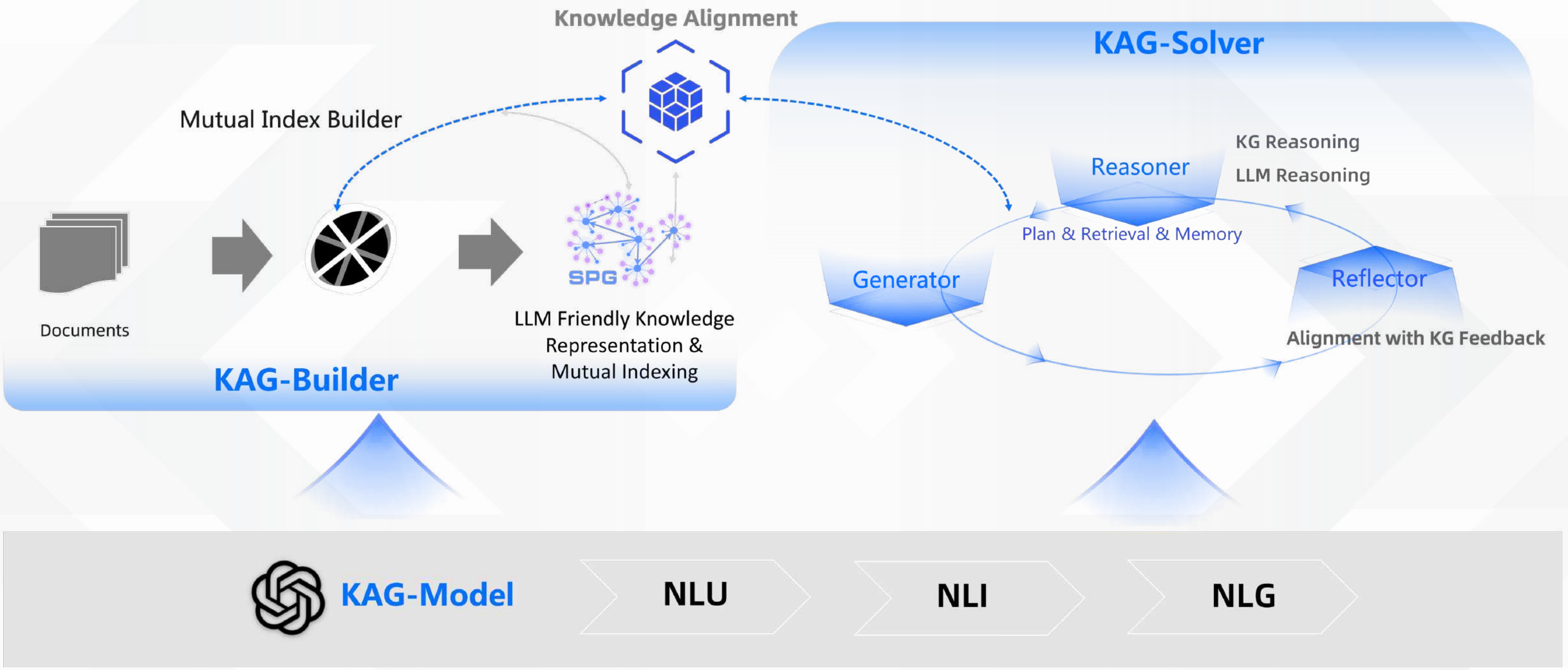
Principles of KAG

Version 0.5

KAG in OpenSPG framework



KAG - Framework

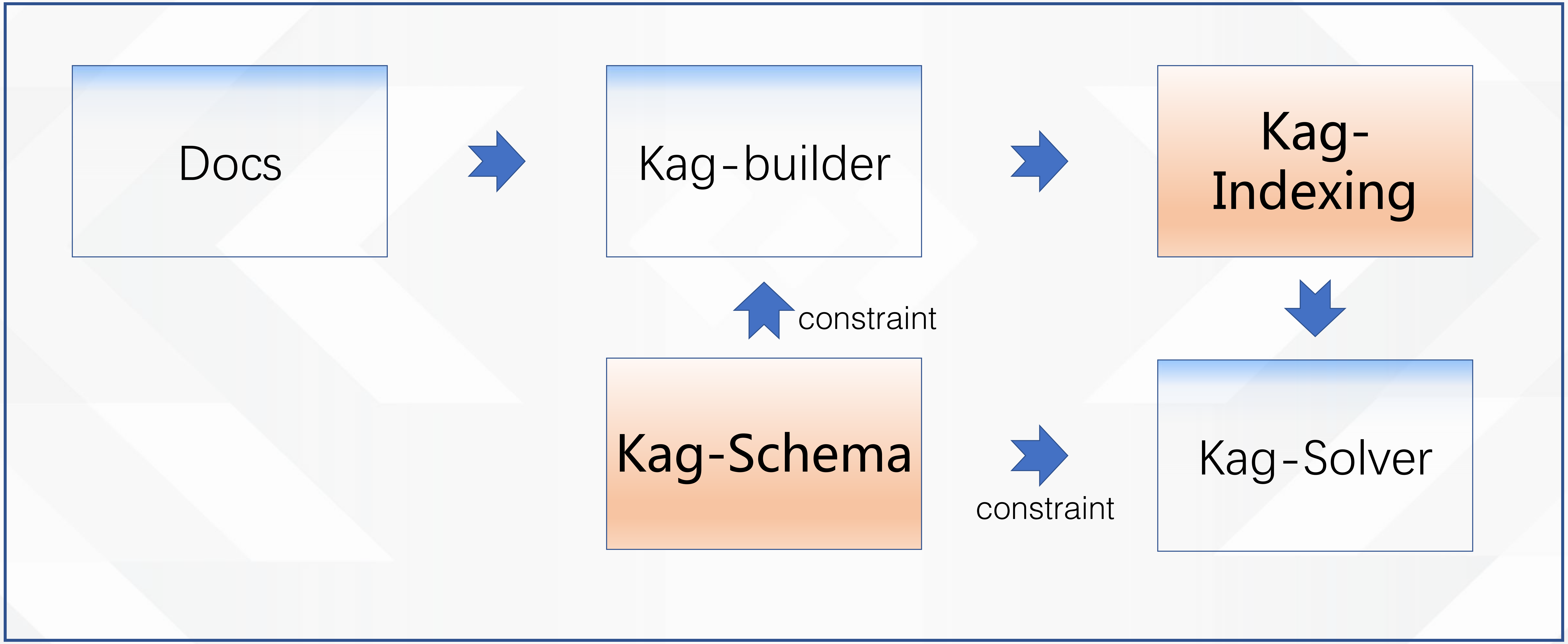


Kag-builder: Construct private domain knowledge into LLM-friendly semantic representation using SPG

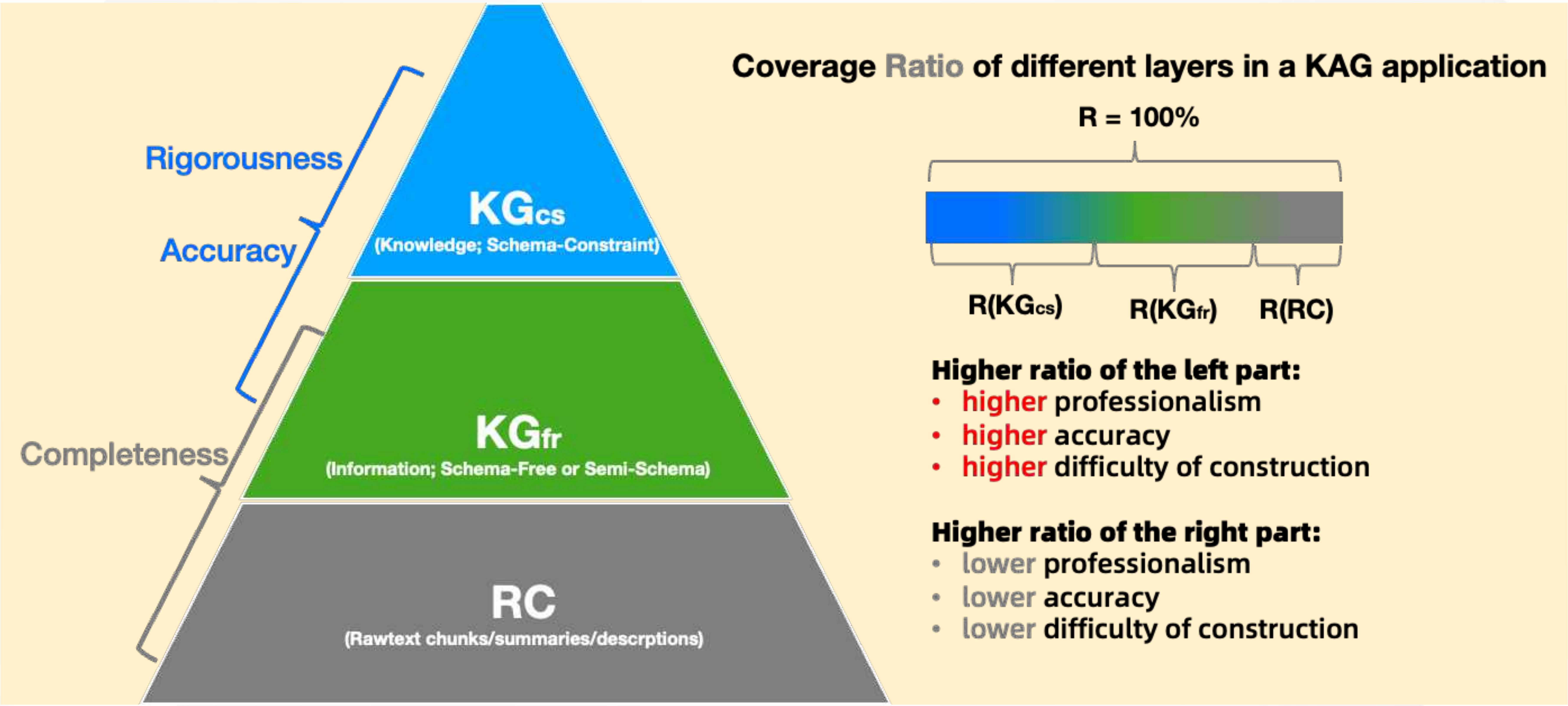
Kag-solver: hybrid reasoning engine guided by logical symbols

Kag-Model: 8B SFT model comparable to 72B model (NLU, NLI, NLG tasks) with significantly reduced resource consumption.

KAG-Schema & Indexing



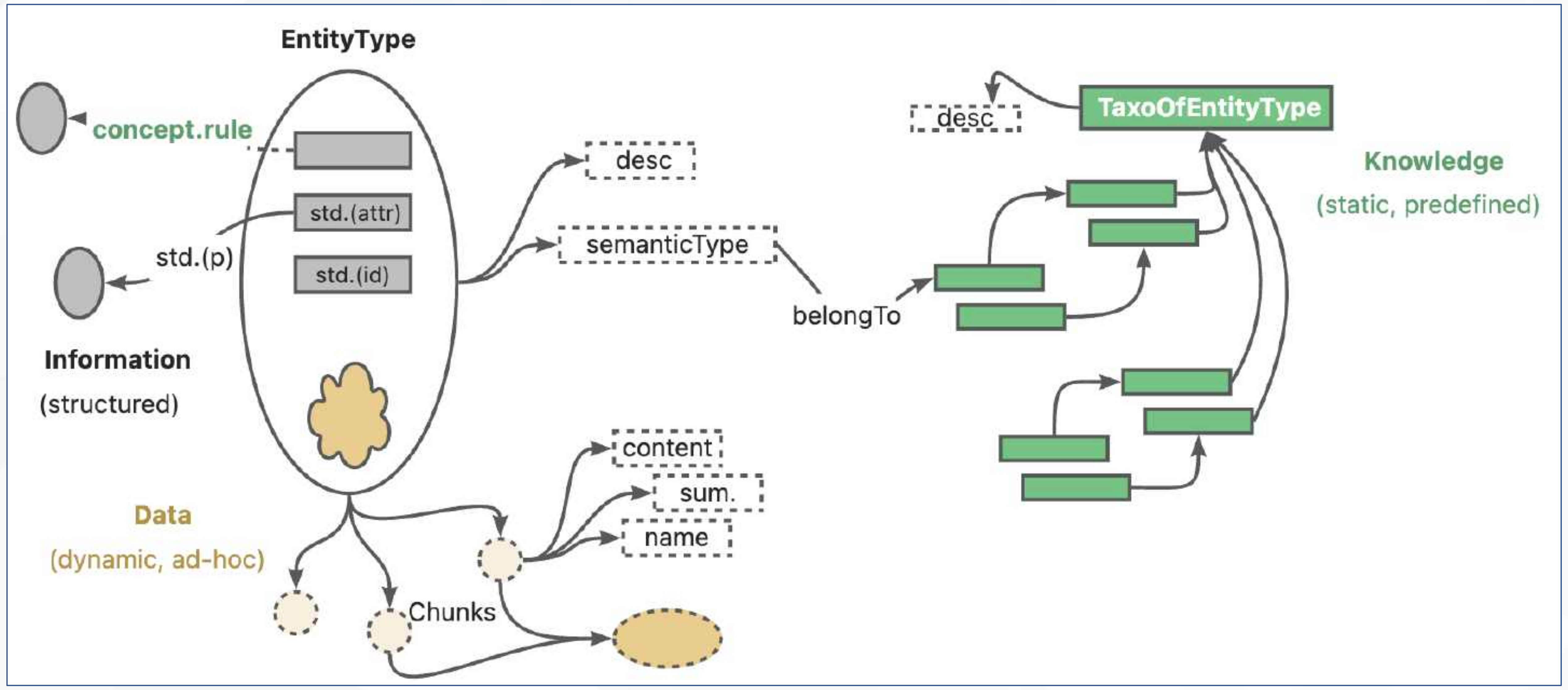
Kag-Indexing



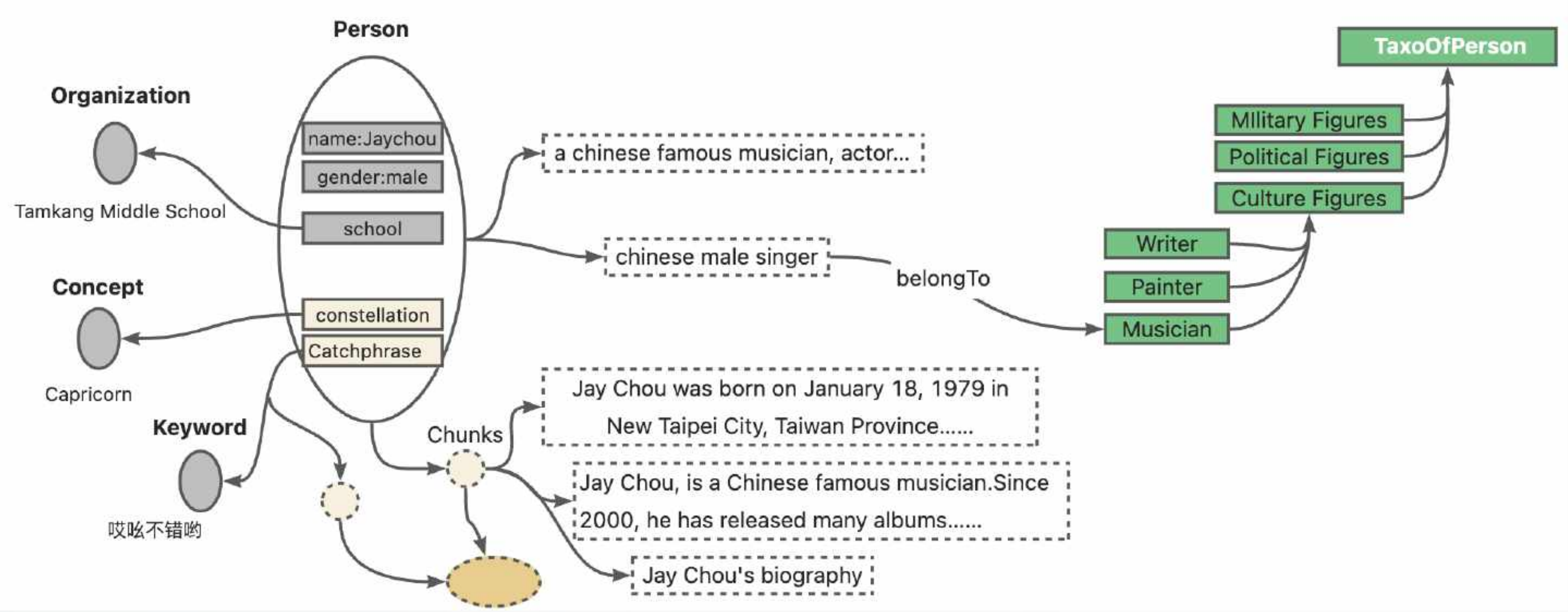
- **LLMFriSPG:** Compatible with Schema-constraint knowledge, Schema-free information, and raw context.
- **Text and graph mutual indexing:** smoothly adjustable in professional decision-making and information retrieval.

LLMFriSPG examples

Kag – Indexing Structure



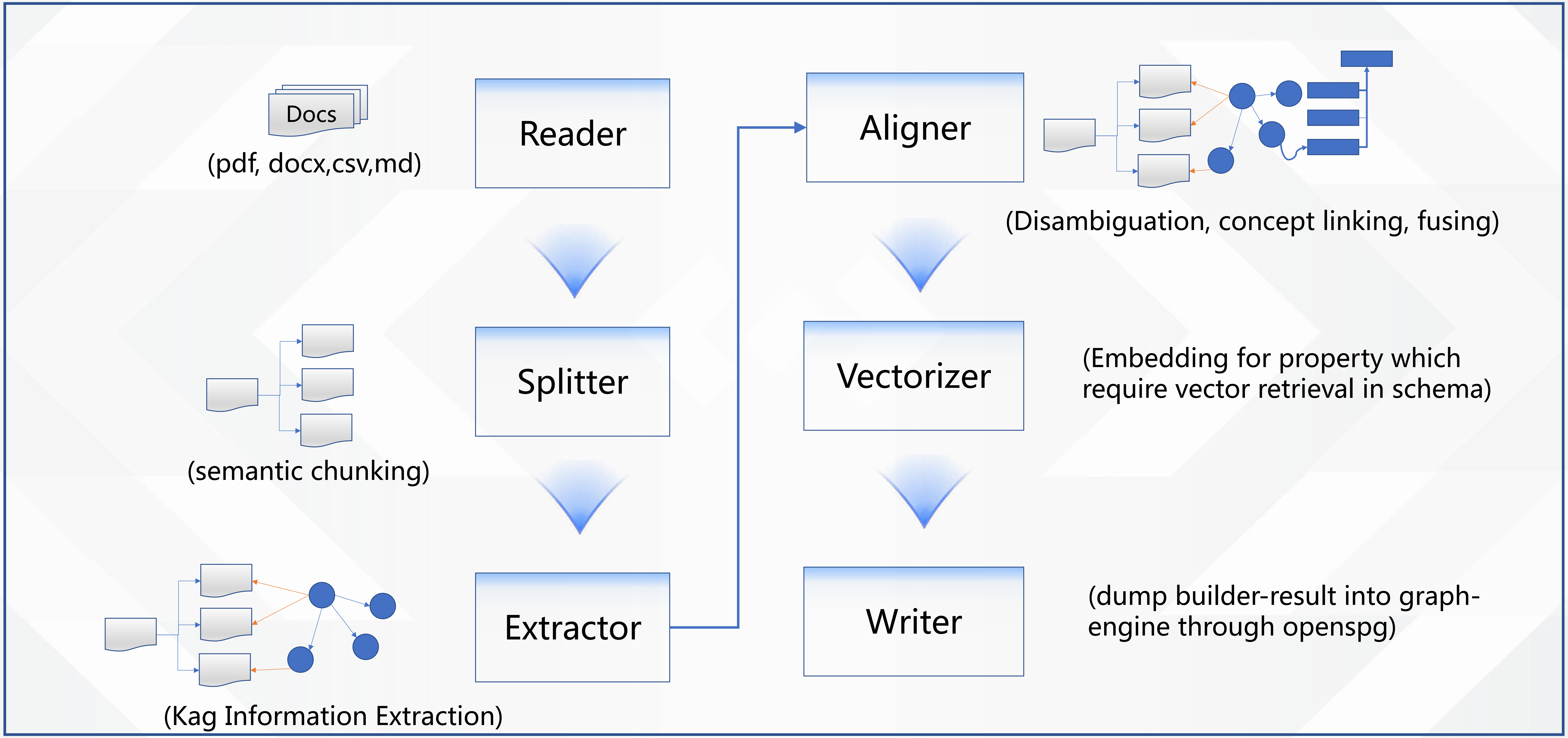
Kag – Indexing instance of Jay Chou



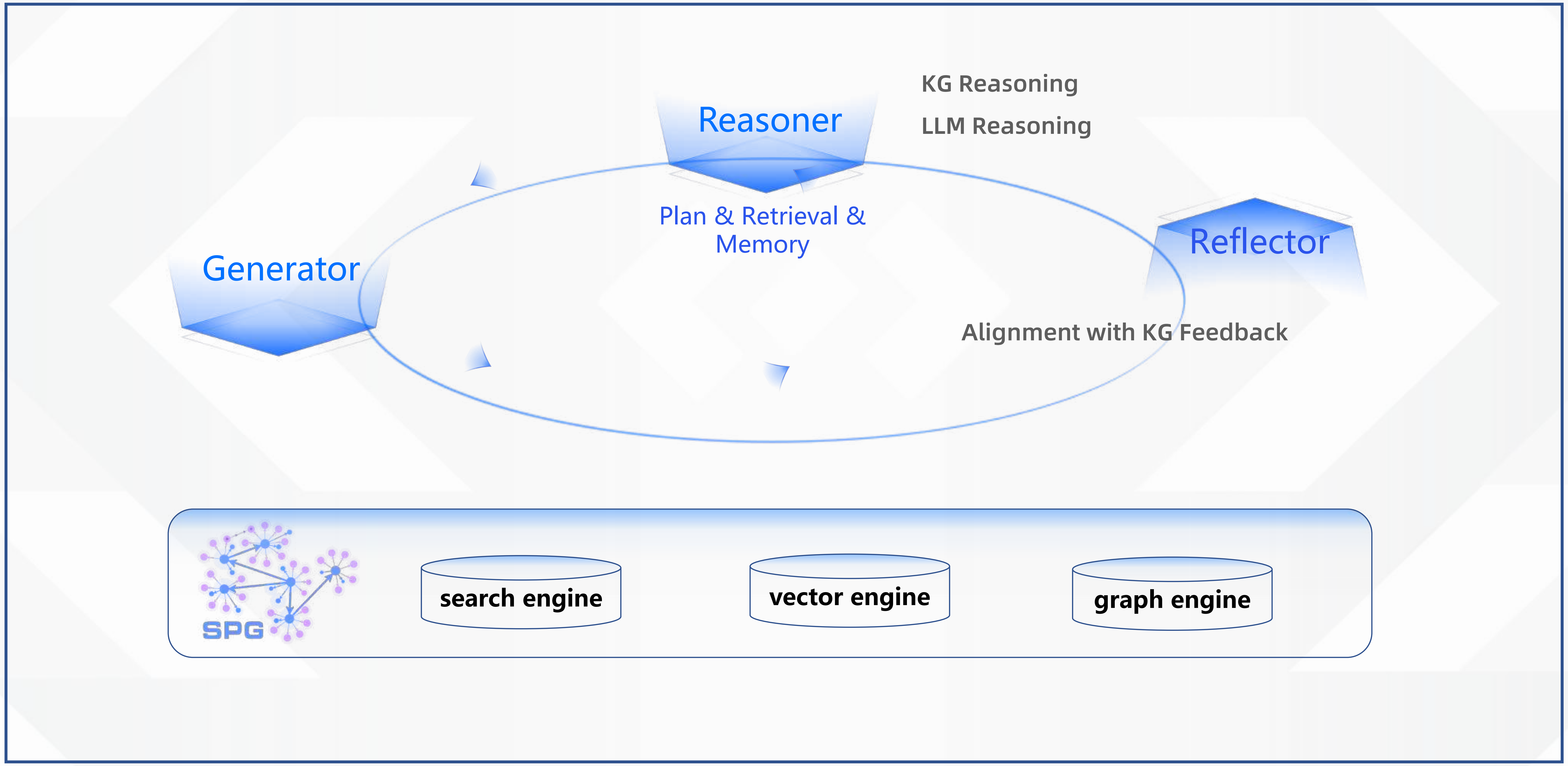
default.schema

```
Organization: EntityType
    properties:
      id: Text
        index: TextAndVector
      name: Text
        index: TextAndVector
      desc: Text
        index: TextAndVector
      semanticType: Text

Person: EntityType
    properties:
      id: Text
        index: TextAndVector
      name: Text
        index: TextAndVector
      desc: Text
        index: TextAndVector
      school: Organization
      gender: Text
      semanticType: Text

Works: EntityType
Concept: EntityType
GeoLocation: EntityType
......
Chunks: EntityType
Others: EntityType
```

Kag-builder

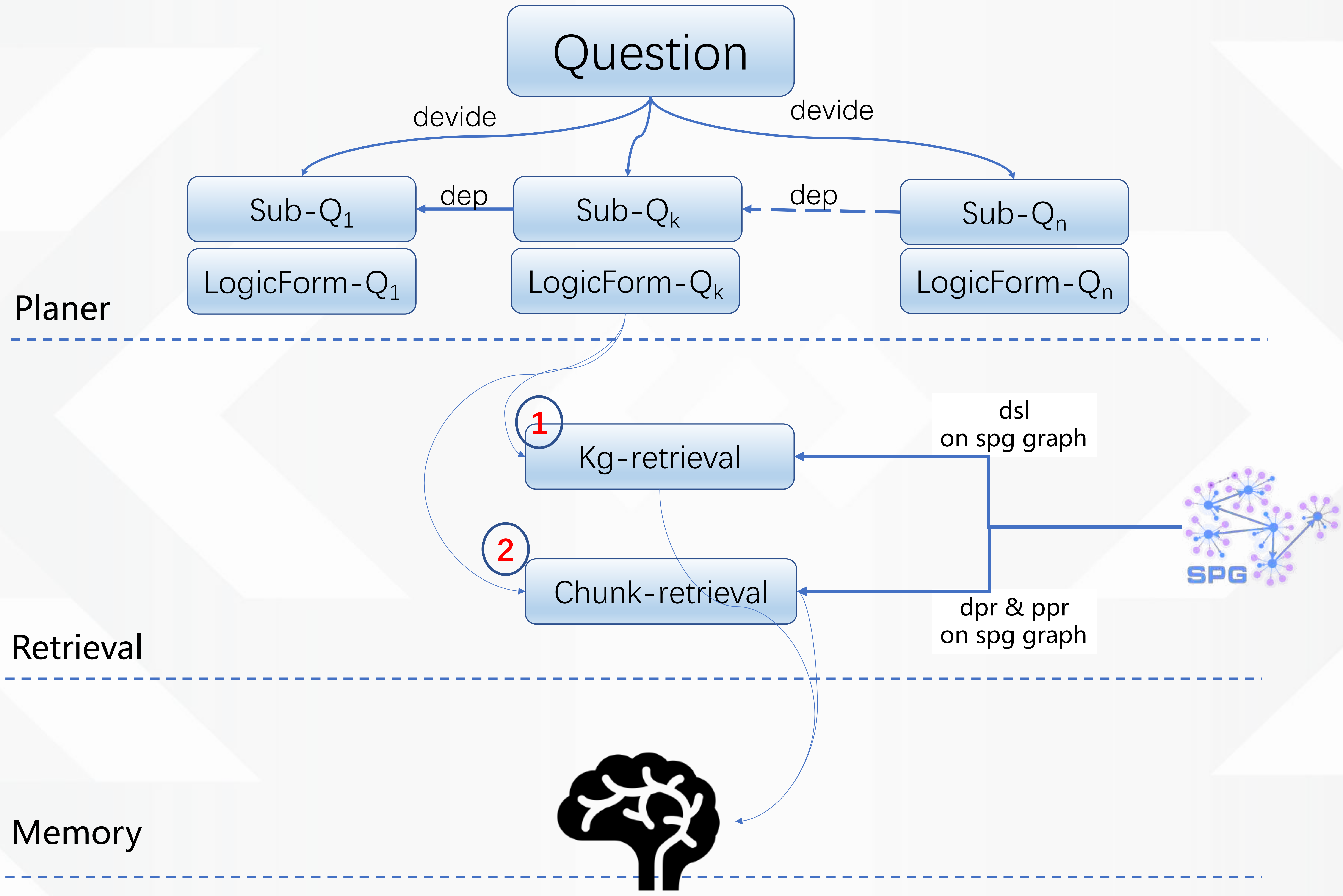

Docs
(pdf, docx,csv,md)
Reader
(semantic chunking)
Splitter
(Kag Information Extraction)
Extractor
Aligner
(Disambiguation, concept linking, fusing)
Vectorizer
(Embedding for property which require vector retrieval in schema)
Writer
(dump builder-result into graph-engine through openspg)

KAG-Solver



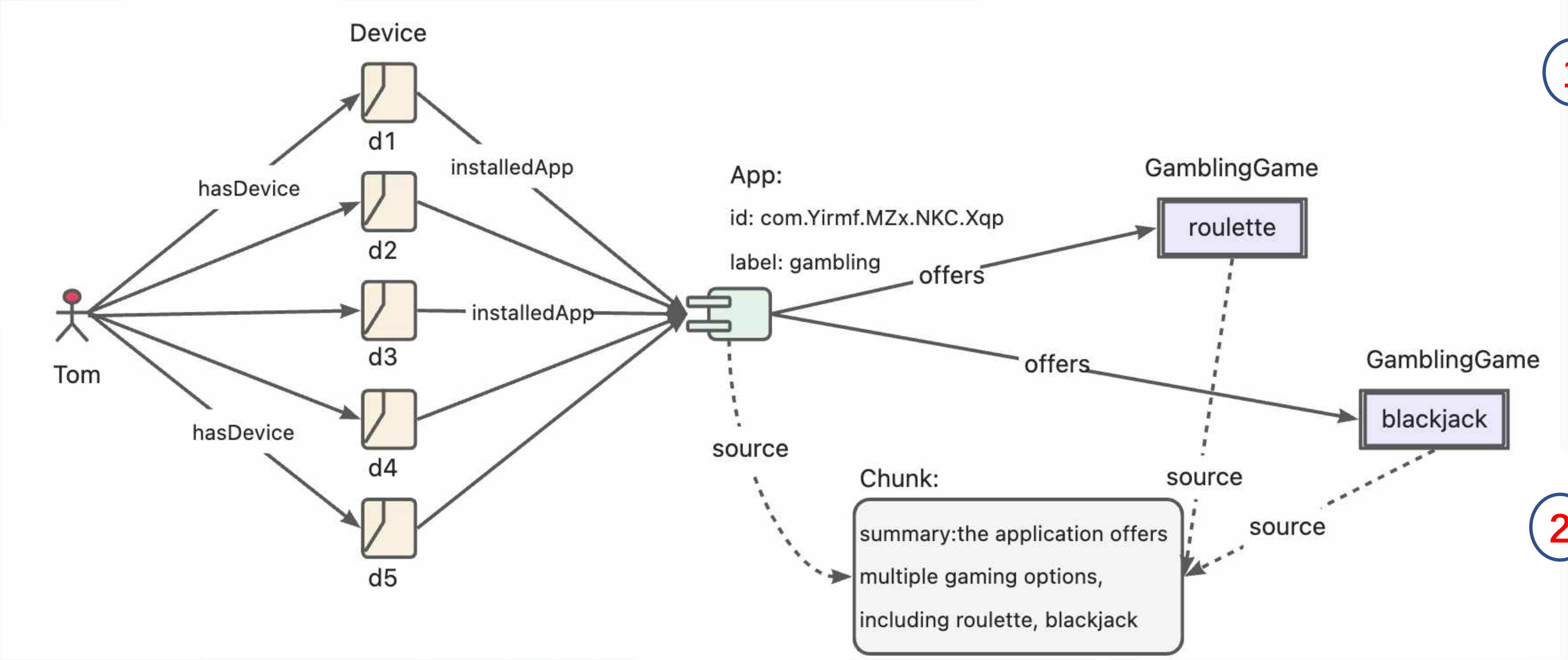
Reasoner
KG Reasoning
LLM Reasoning
Plan & Retrieval & Memory
Generator
Reflector
Alignment with KG Feedback
SPG
search engine
vector engine
graph engine

reasoner of Kag-Solver



KAG's rigorous decision-making in the risk-mining

RC & KG_{fr} & KG_{cs}



1 define riskAppTaxo rule

Plain Text

```
Define (s:App)-[p:belongTo]->(o:`TaxOfRiskApp`/`GamblingApp`) {
    Structure {
        (s)
    }
    Constraint {
        R1("risk label marked as gambling") s.riskMark like "%Gambling%"
    }
}
```

2 define app developper rule

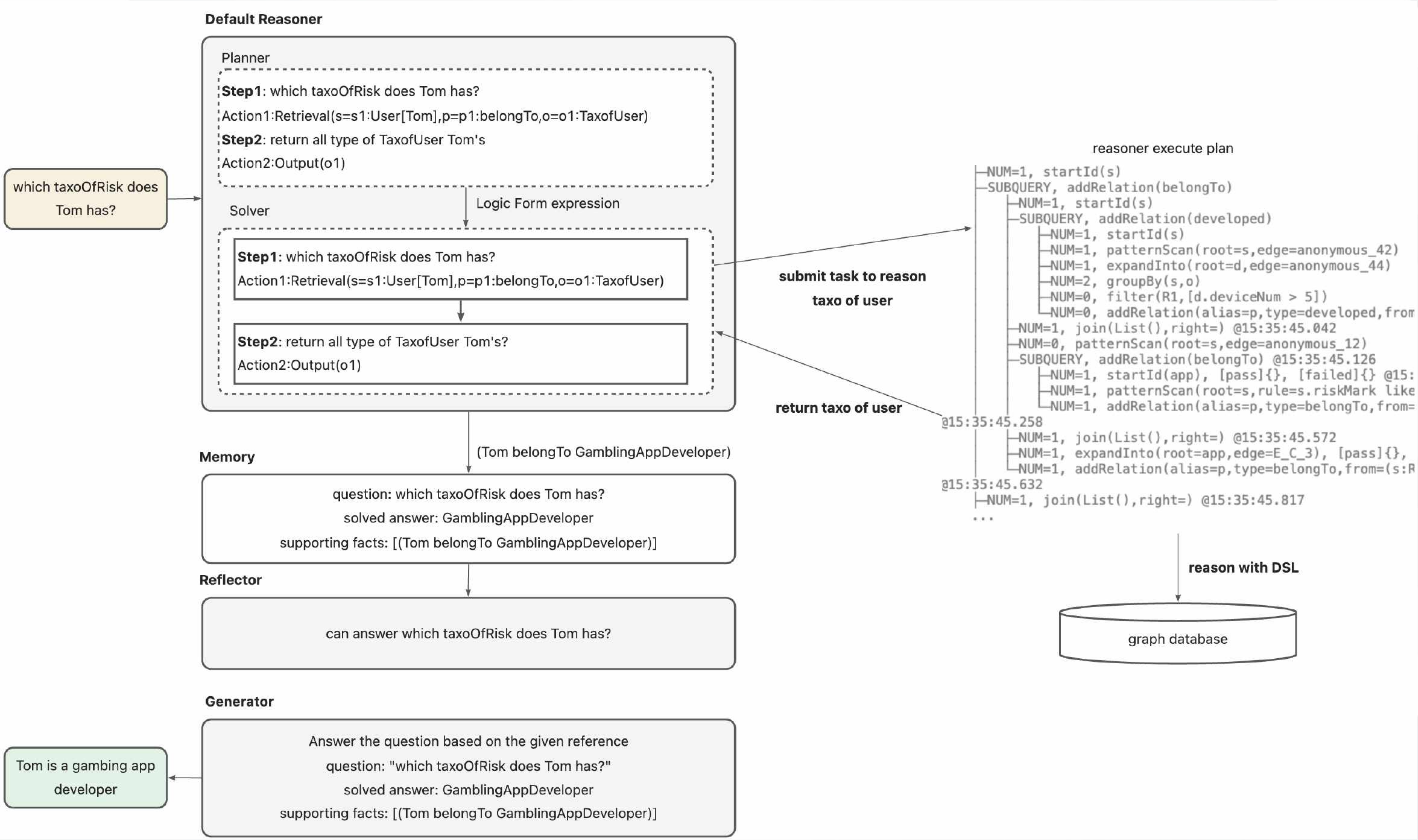
```
Define (s:Person)-[p:developed]->(o:App) {
    Structure {
        (s)-[:hasDevice]->(d:Device)-[:install]->(o)
    }
    Constraint {
        deviceNum = group(s,o).count(d)
        R1("device installed same app"): deviceNum > 5
    }
}
```

3 define a RiskUser of gambling app rule

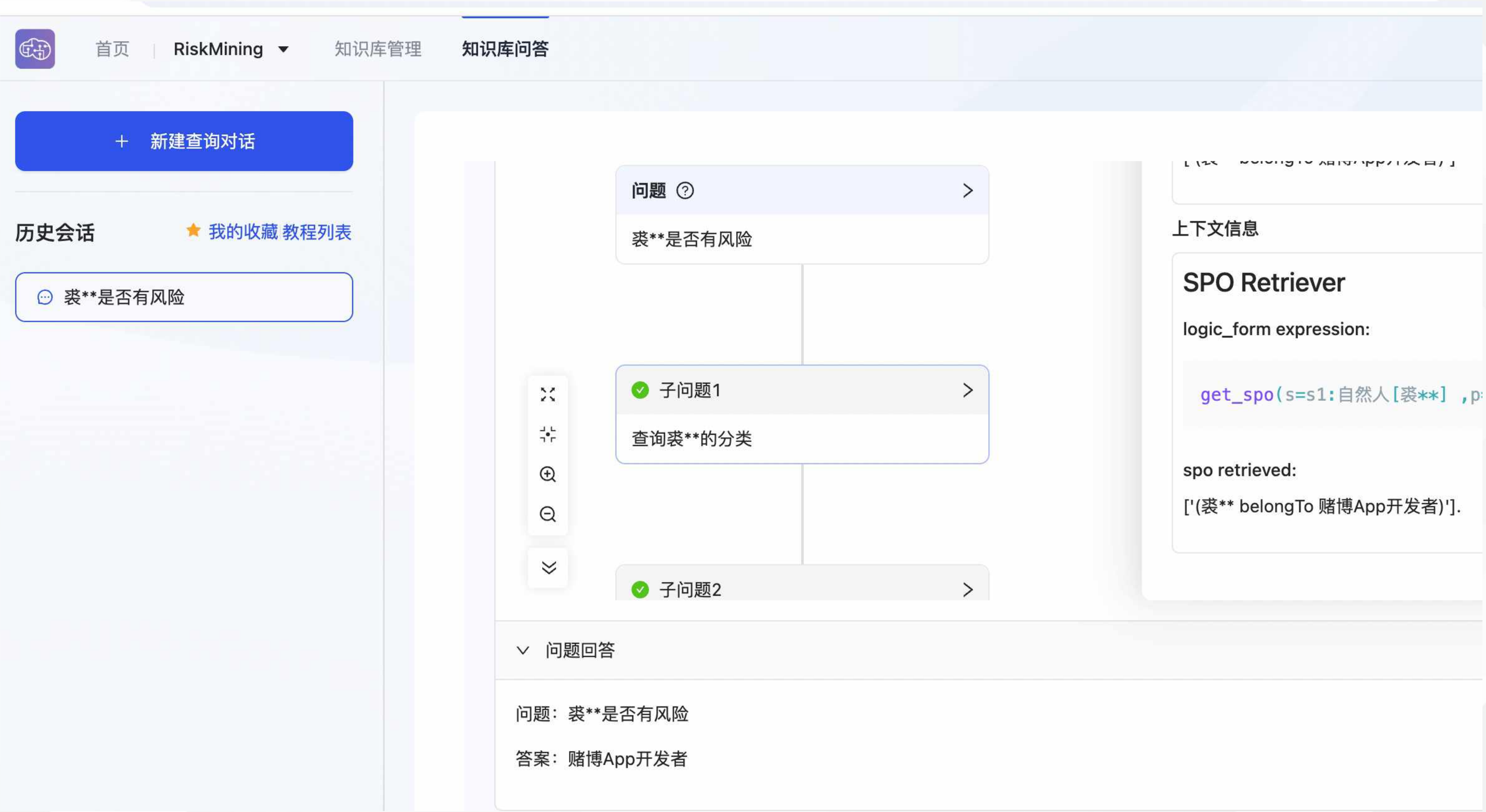
Plain Text | 复制代

```
Define (s:Person)-[p:belongTo]->(o:`TaxOfRiskUser`/`DeveloperOfGamblingApp`) {
    Structure {
        (s)-[:developed]->(app:`TaxOfRiskApp`/`GamblingApp`)
    }
    Constraint {
    }
}
```

KAG's rigorous decision-making in the risk-mining



Kag-Solver decision making result



Deploy & Use

Product Mode

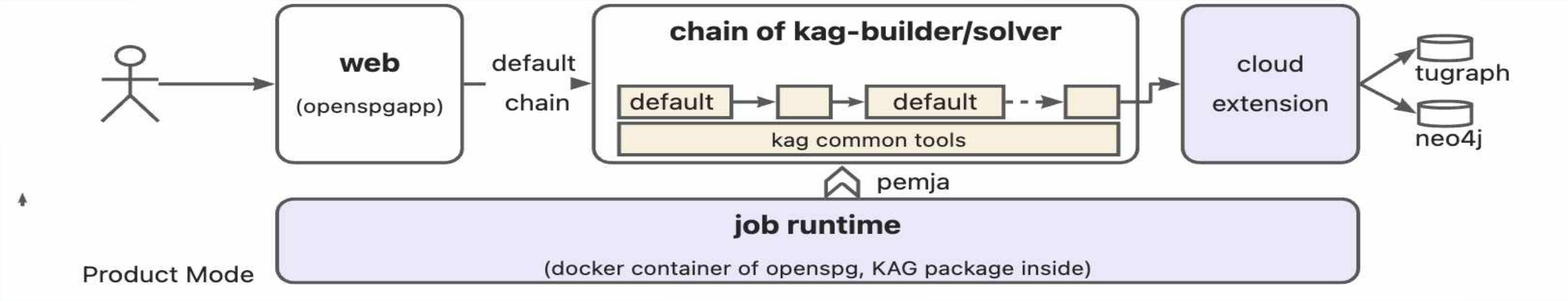
Developer Mode

蚂蚁开源 ANT OPEN SOURCE

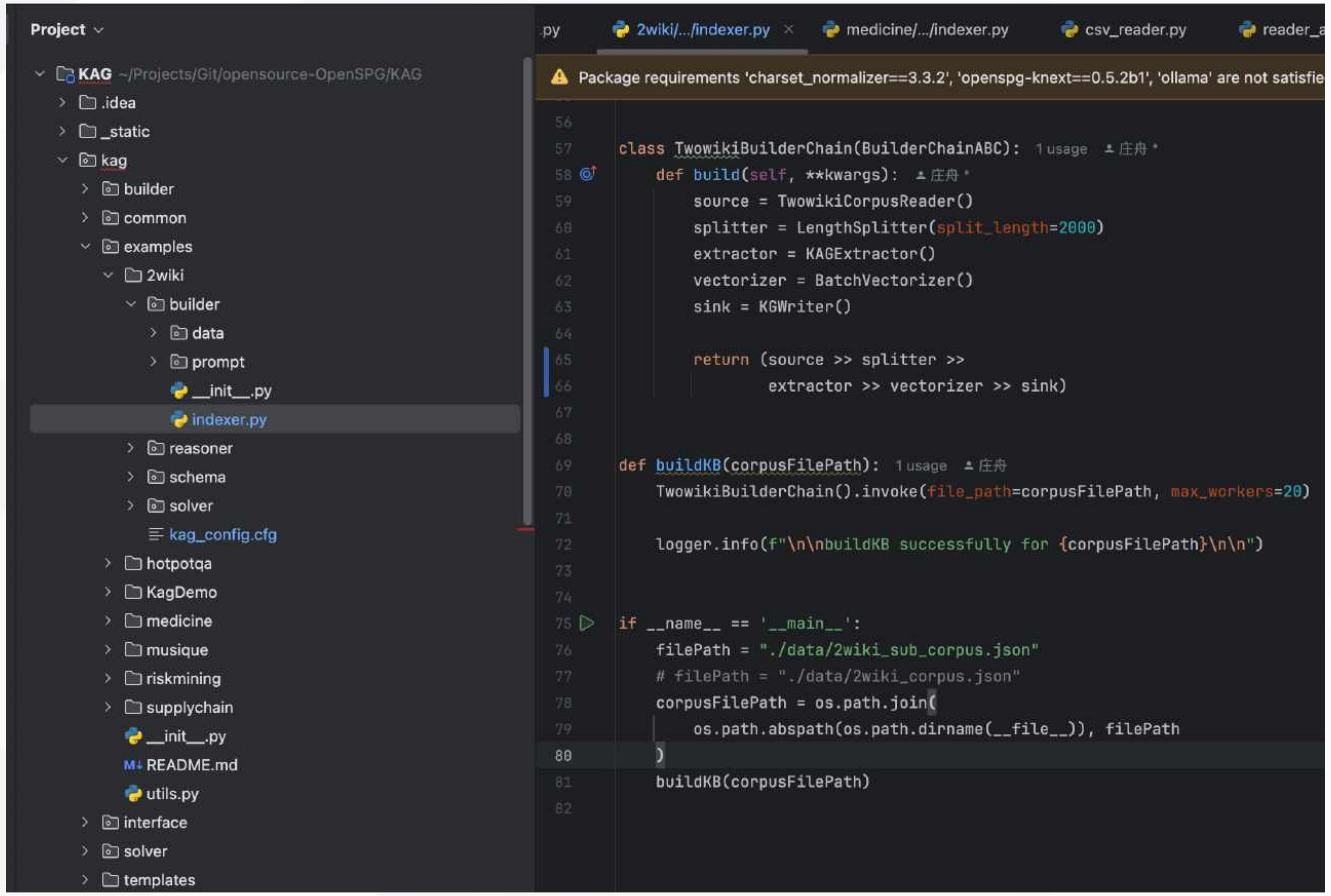
KAG usage（Product Mode）

Product Mode : readme

- kag default schema + default builder chain
- Openspg provides runtime env for Kag-builder chain through pemja
- kag-builder call API of openspg server for dumping extraction result to graph-engine

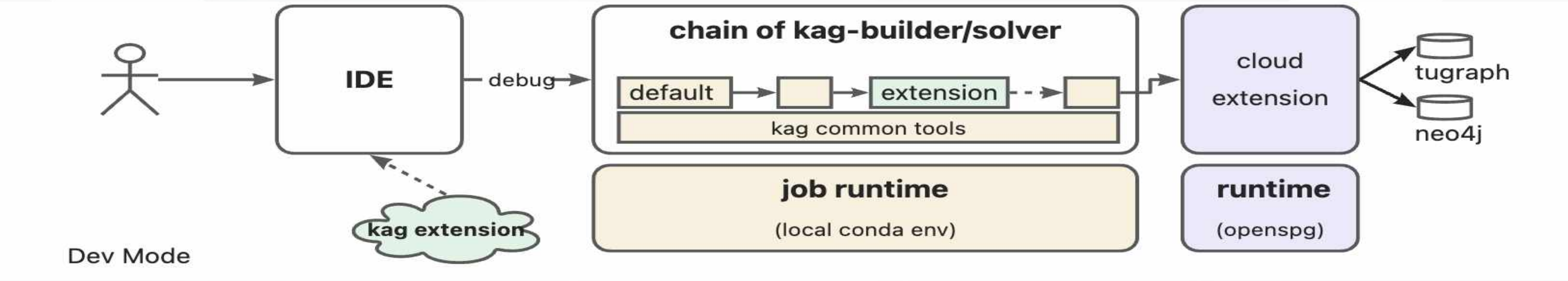


KAG usage（Developer Mode）



Developer Mode：readme

- developer customize schema & builder chain
- Local python IDE provides runtime env for Kag-builder chain
- kag-builder call API of openspg for dumping extraction result to graph-engine



KAG applications

KAG built-in examples

```
kag
├──── examples
│     ├──── 2wiki
│     ├──── hotpotqa
│     ├──── medicine
│     ├──── musique
│     ├──── riskmining
│     │     ├──── builder
│     │     │     ├──── data
│     │     │     └──── indexer.py
│     │     ├──── kag_config.cfg
│     │     ├──── reasoner
│     │     │     ├──── client.py
│     │     │     └──── gambling_app.dsl
│     │     ├──── schema
│     │     │     ├──── RiskMining.schema
│     │     │     └──── concept.rule
│     │     └──── solver
│     │           ├──── prompt
│     │           └──── qa.py
│     │
│     ├──── supplychain
```

Framework	Model	HotpotQA		2WikiMultiHopQA		MuSiQue	
		EM	F1	EM	F1	EM	F1
NativeRAG [24, 23]	ChatGPT-3.5	43.4	57.7	33.4	43.3	15.5	26.4
HippoRAG [6, 23]	ChatGPT-3.5	41.8	55.0	46.6	59.2	19.2	29.8
IRCoT+NativeRAG	ChatGPT-3.5	45.5	58.4	35.4	45.1	19.1	30.5
IRCoT+HippoRAG	ChatGPT-3.5	45.7	59.2	47.7	62.7	21.9	33.3
IRCoT+HippoRAG	DeepSeek-V2	51.0	63.7	48.0	57.1	26.2	36.5
KAG (ours)	DeepSeek-V2	**62.5**	**76.2**	**67.8**	**76.7**	**36.7**	**48.7**

Table 9: The end-to-end generation performance of different RAG models on three multi-hop question answering datasets. Bold text indicates that the same base model performs best. NativeRAG and HippoRAG use single-step retrieval, while other models employ multi-step retrieval.

	Retriever	HotpotQA		2WikiMultiHopQA		MuSiQue	
		Recall@2	Recall@5	Recall@2	Recall@5	Recall@2	Recall@5
Single-step	BM25 [25]	55.4	72.2	51.8	61.9	32.3	41.2
	Contriever [26]	57.2	75.5	46.6	57.5	34.8	46.6
	GTR [27]	59.4	73.3	60.2	67.9	37.4	49.1
	RAPTOR [28]	58.1	71.2	46.3	53.8	35.7	45.3
	Proposition [29]	58.7	71.1	56.4	63.1	37.6	49.3
	NativeRAG [24, 23]	64.7	79.3	59.2	68.2	37.9	49.2
	HippoRAG [6, 23]	60.5	77.7	70.7	89.1	40.9	51.9
Multi-step	IRCoT + BM25	65.6	79.0	61.2	75.6	34.2	44.7
	IRCoT + Contriever	65.9	81.6	51.6	63.8	39.1	52.2
	IRCoT + NativeRAG	67.9	82.0	64.1	74.4	41.7	53.7
	IRCoT + HippoRAG	67.0	83.0	**75.8**	**93.9**	45.3	57.6
	KAG (ours)	**72.8**	**88.8**	65.4	91.9	**48.5**	**65.7**

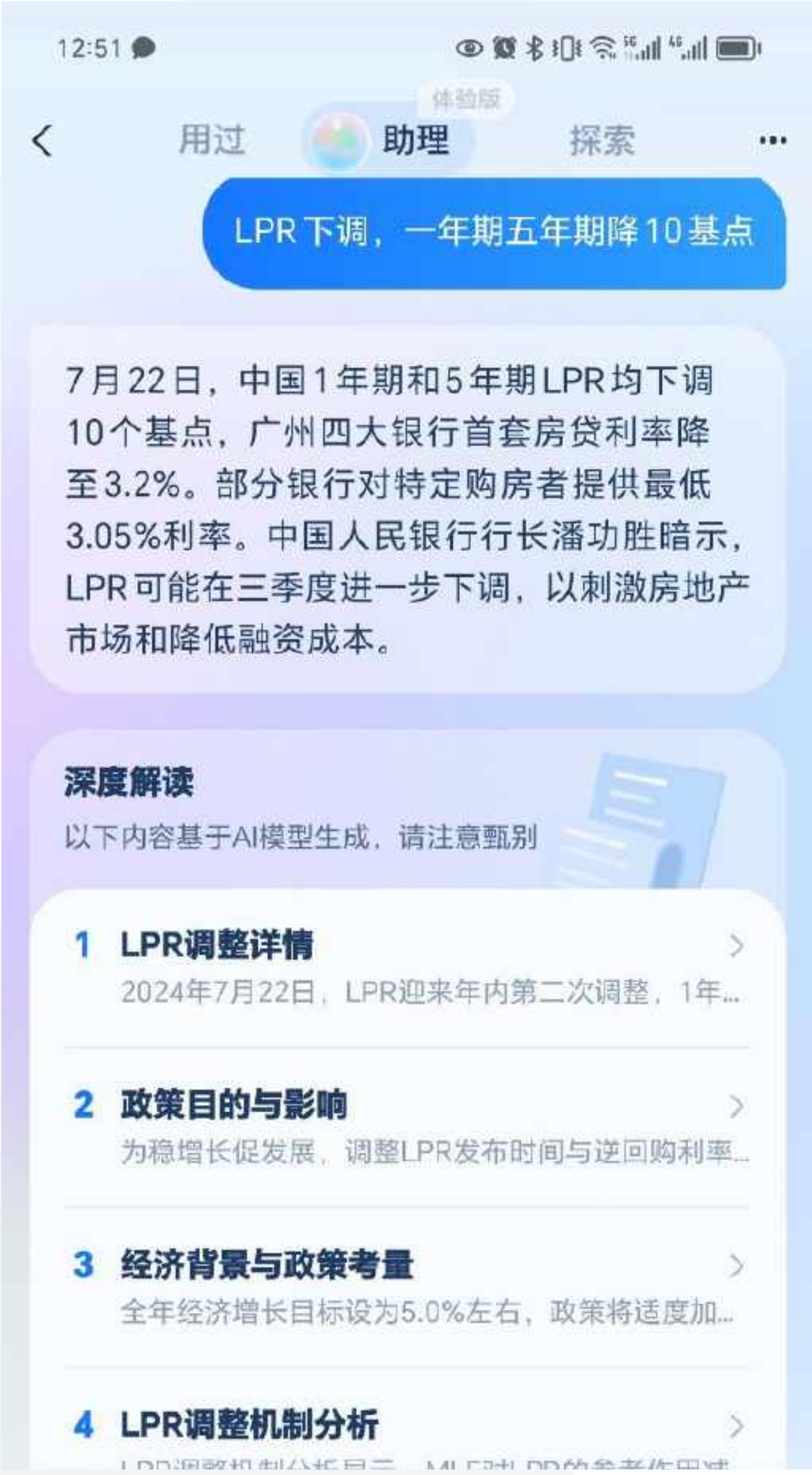
Table 10: The performance of different retrieval models on three multi-hop question-answering datasets.

- Reproduction of KAG examples, please refer to: kag user manual

蚂蚁开源
ANT OPEN SOURCE

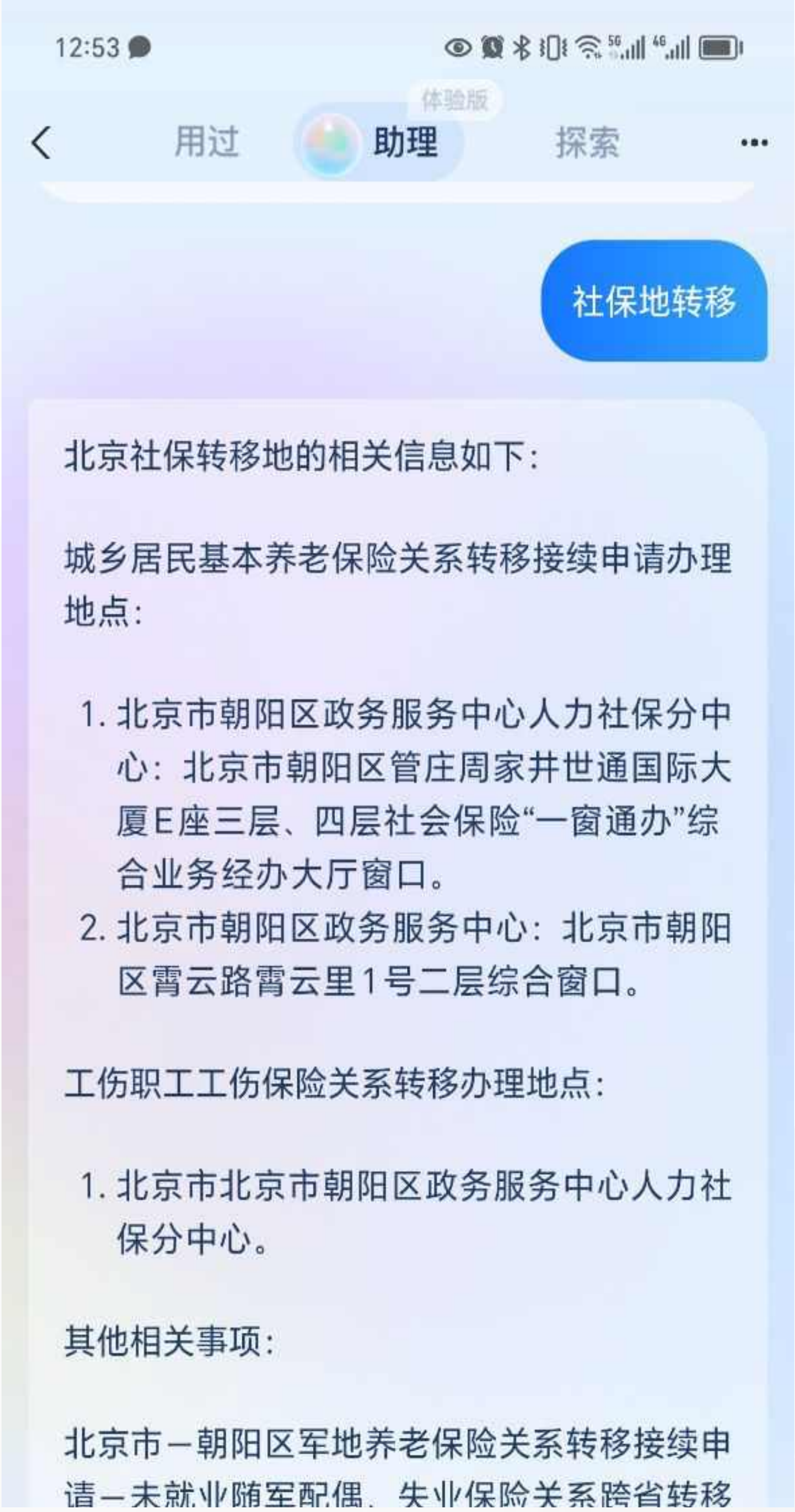
KAG applications in AntGroup

Analytical Writing

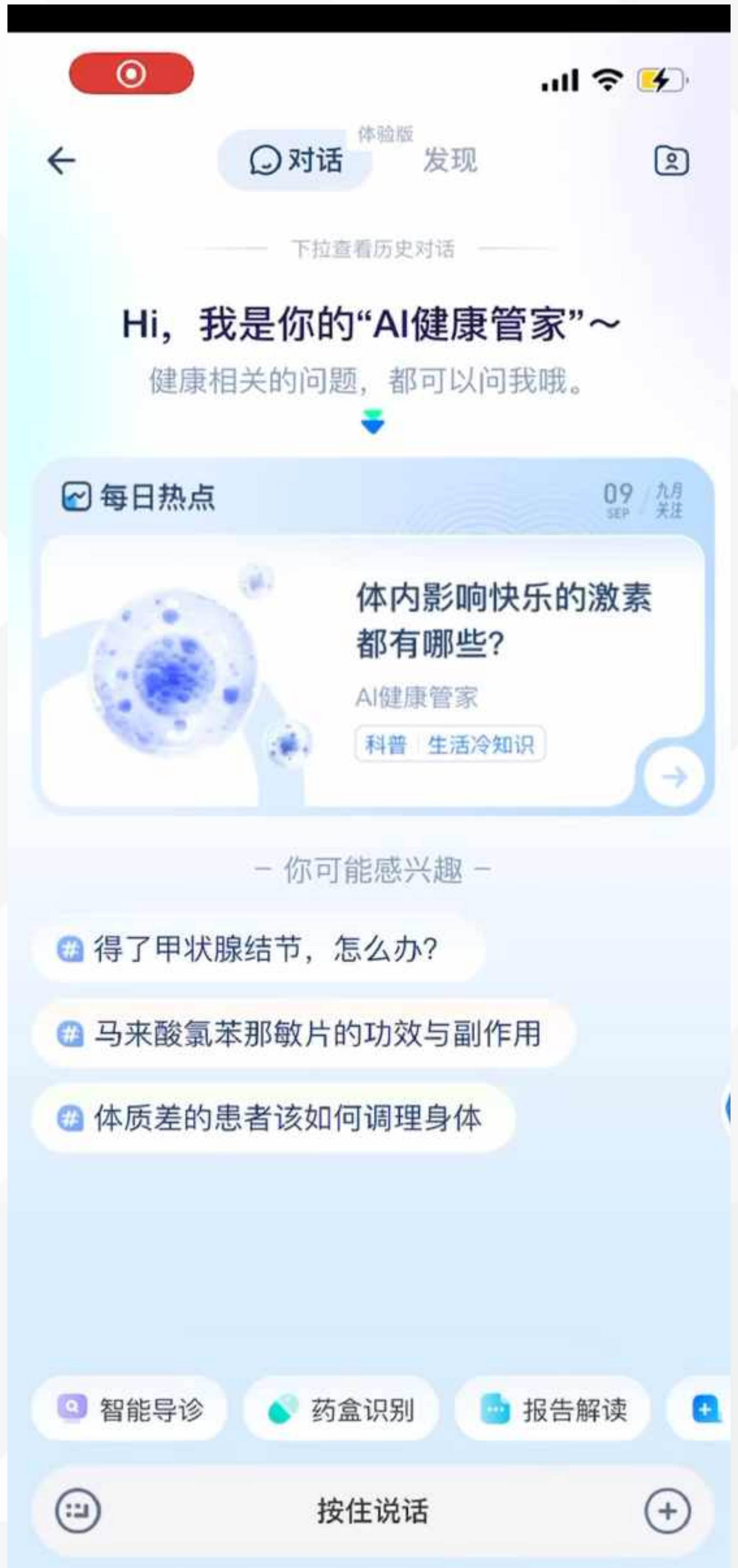


Factual QA

Government Service Q&A



Healthcare QA



Future Plans

蚂蚁开源
ANT OPEN SOURCE

OpenSPG-KAG future plans

Modules	Capability Upgrade Items	Release Schedule
Kag-web	1、schema customize 2、interactive data retrieval	Refer to openspg official website
Kag-builder	1、domain data injection 2、distributed version	
Kag-solver	1、Logical-Form completeness 2、QFS tasks, dialogue QA	
Kag-model	1、kag model release	

Contact Us

KAG: https://github.com/OpenSPG/KAG

KAG User manual: ReadMe

OpenSPG official site: https://spg.openkg.cn/en-US

蚂蚁开源
ANT OPEN SOURCE

Thanks & QA