AGENTICT^{2}S: Robust Text-to-SPARQL via Agentic Collaborative Reasoning over Heterogeneous Knowledge Graphs

Anonymous ACL submission

Abstract

Question answering over heterogeneous knowledge graphs (KGQA) requires reasoning across diverse schemas, partial alignments, and distributed endpoints. Existing text-to-SPARQL methods either depend on domain-specific fine-tuning or assume a single graph, limiting transfer to low-resource domains and hindering cross-graph queries. We present AGENTICT^{2}S, a modular framework that decomposes a question into subgoals handled by agents for retrieval, SPARQL generation, and verification. An allocator selects each subgoal to one or more candidate graphs using weak-to-strong schema alignment, and reduces to the single endpoint in the standard single-KG setting. To improve reliability, a two-stage verifier filters structurally invalid and semantically underspecified queries via symbolic validation and counterfactual consistency checks. Across three Wikidata benchmarks and a circular economy multi-KG benchmark, AGENTICT^{2}S improves execution accuracy and triple-level F_1 by +21.28 and +22.83 percentage points, respectively, on average over the strongest baseline (AutoGen). On the heterogeneous circular economy benchmark, it reduces average input tokens by 46.4% (875 vs. 1632 ATU). Overall, the results show that agentic decomposition, schema-aware routing, and explicit verification yield more reliable text-to-SPARQL for heterogeneous, multi-graph KGQA.

1 Introduction

Large language models (LLMs) demonstrate strong performance in a range of natural language processing tasks (Min et al., 2023), but face limitations in knowledge-intensive settings. Because their outputs are based on pre-trained parameters, LLMs are prone to hallucination and may lack factual consistency (Ji et al., 2023). Retrieval-augmented generation (RAG) addresses this by incorporating external information at inference time (Lewis et al., 2020), although most RAG systems operate over unstructured text, which lacks the semantic structure required for accurate multi-hop reasoning.

Structured retrieval methods such as GraphRAG (Ji et al., 2024) have been proposed to integrate LLMs with knowledge graphs (KGs), which offers improved precision and interpretability. However, existing systems often embed large subgraphs in prompts (Cui et al., 2024), increasing latency and exceeding context limits; rely on weakly supervised retrievers (Yasunaga et al., 2021); and couple retrieval, generation and validation into single-agent pipelines (He et al., 2024; Xu et al., 2024), reducing modularity and adaptability. In addition, even state-of-the-art models struggle to generate semantically correct SPARQL queries, particularly on unseen graphs (Meyer et al., 2024). SPARQL is a standardized query language for RDF-based KGs (W3C, 2013), used to express structured queries involving logical conditions, joins, and filters.

This work introduces AGENTICT^{2}S, a modular framework for answering questions on heterogeneous KGs. The system decomposes a natural language query into sub-goals corresponding to reasoning operations such as entity lookup or comparison. Each subgoal is routed to candidate graphs via weak-to-strong schema alignment and retrieval evidence, translated into executable SPARQL using schema-guided decoding, and validated using symbolic checks and counterfactual consistency analysis. The architecture separates retrieval, generation, and verification into distinct agents to improve transparency, error isolation, and generalizability.

To our knowledge, AGENTICT^{2}S is among the first approaches to cast heterogeneous, multi-graph KGQA text-to-SPARQL as a multi-agent pipeline. The system comprises five agents—a subgoal decomposer, graph allocator, SPARQL synthesizer, dual-stage verifier, and result integrator—that coordinate to support compositional reasoning and reliable query execution. We evaluate the design on

Wikidata benchmarks and a circular economy setting, where distributed data sources and non-unified classification schemes motivate schema-aware routing and verifiable synthesis (Atiku, 2020).

Contributions.

- We formulate heterogeneous, multi-graph KGQA as a text-to-SPARQL planning problem, where a question is decomposed into subgoals that yield a set of executable SPARQL subqueries and an integrated final answer, covering single-KG QA as a special case.
- We propose AGENTICT^2S, a modular agent pipeline with (i) weak-to-strong schema alignment for per-subgoal graph allocation, (ii) schema-grounded SPARQL synthesis with template scaffolding and post-hoc repair, and (iii) a dual-stage verifier that combines symbolic checks with counterfactual consistency testing.
- We evaluate on three Wikidata text-to-SPARQL benchmarks and a circular economy multi-KG benchmark (328 questions over three in-house RDF KGs), using a matched-backbone zero-shot protocol and component ablations to quantify the contribution of each module.

2 Related Work

RAG methods incorporate external information to improve the factual consistency of the language model output (Lewis et al., 2020; Gao et al., 2023). While effective for unstructured question answering, they assume flat input formats and are not well suited for tasks involving structured queries, multi-hop reasoning, or schema constraints.

GraphRAG (Edge et al., 2024; Wu et al., 2025a) and related methods extend RAG by converting local graph neighborhoods into textual prompts. These methods retain some structural context but are limited by prompt length, formatting dependencies, and an assumption of a single underlying graph. Most also lack validation mechanisms for generated queries (Wang et al., 2025).

Other work decomposes question answering into modular components for retrieval, reasoning, and execution (Tran et al., 2025; Barbosa et al., 2024). Multi-agent systems extend this idea by assigning subtasks to separate agents (Guo et al., 2024; Yang et al., 2025). For example, ReAgent (Xinjie et al., 2025) applies local and global backtracking in multi-hop QA over text. These systems are typically applied to unstructured input and do not address structured query generation.

General purpose agent frameworks such as ReAct (Yao et al., 2023), AutoGen (Wu et al., 2024), CAMEL (Li et al., 2023), and MetaGPT (Hong et al., 2023) implement role-based decomposition for planning and tool interaction across tasks. These methods have been applied to information retrieval, decision-making, and code synthesis in unstructured or semi-structured settings. They are not designed for structured KGQA and lack support for schema-aware reasoning or query verification.

Positioning. AGENTICT^2S targets robust text-to-SPARQL over heterogeneous, multi-graph KGs. It combines (i) subgoal decomposition for compositional queries, (ii) schema-aware routing to select compatible graphs per subgoal, (iii) schema-grounded synthesis with template scaffolding and post-hoc repair, and (iv) explicit verification to filter invalid or underspecified queries before execution. This design improves executability and accuracy without task-specific fine-tuning.

3 Problem Formulation

Given a natural language question q, the task is to produce a set of executable SPARQL subqueries $\{s_i\}_{i=1}^{k}$ (optionally aligned with subgoals $\{g_i\}_{i=1}^{k}$) that retrieve a complete and semantically correct final answer A by querying an appropriate subset of a heterogeneous KG collection $\mathcal{G} = \{G_1, G_2, \ldots, G_n\}$. This formulation also covers the standard single-KG (single-endpoint) setting as a special case where $n = 1$. Each KG $G_i \in \mathcal{G}$ consists of entities and relations represented as triples (h, r, t), where h and t are entities, and r is a relation type (Saxena et al., 2020).

A valid subquery s_i is expected to satisfy two properties. First, it must be executable, meaning it can be evaluated on its allocated graph $G_i \in \mathcal{G}$ without syntax or schema violations. Second, it must be semantically accurate, in that it reflects the intent of q (or the corresponding subgoal) and returns a correct intermediate answer set $A_i \subseteq \mathrm{Ent}$, where Ent is the union of entity sets across all graphs in $\mathcal{G}$. The final answer A is obtained by integrating intermediate results $\{A_i\}_{i=1}^{k}$.

This task presents several challenges. The graphs in $\mathcal{G}$ may differ in schema, vocabulary, and structural representation, introducing substantial het-

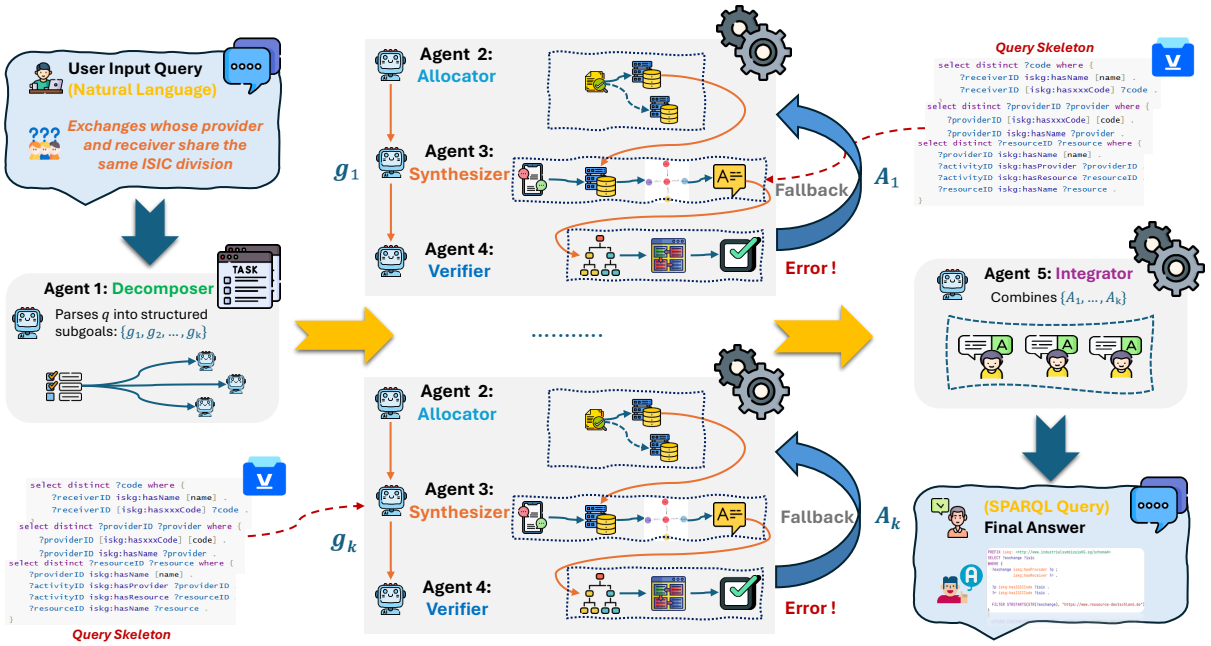


Figure 1: Overview of the AGENTICT^{2}S framework. The system decomposes a natural language query into subgoals, allocates them to relevant graphs, generates SPARQL queries, verifies their correctness, and integrates the answers.

erogeneity. Answering a question may require distributed reasoning, where information must be aggregated from multiple KGs. Furthermore, many KGQA benchmarks lack the gold-standard SPARQL annotations, limiting the feasibility of supervised learning. Finally, practical systems must meet efficiency requirements, making retrieval and query generation time sensitive.

The objective is to develop a system that, given an input question q and a collection $\mathcal{G}$ of KGs, produces executable subqueries $\{s_i\}_{i=1}^{k}$ such that executing them on an appropriate subset of $\mathcal{G}$ yields intermediate results that can be integrated into a correct and complete final answer A. Table 6 in Appendix G summarizes notations used.

4 System Design

We present AGENTICT^{2}S, a modular multi-agent framework to answer natural language questions on heterogeneous KGs. Unlike end-to-end GraphRAG pipelines (Han et al., 2024), which tightly link retrieval and generation in a single prompt-based step, AGENTICT^{2}S separates the task into distinct components: subgoal decomposition, adaptive retrieval, query generation, verification, and answer aggregation. Each component is implemented as an independent agent with a clear input-output interface (Wu et al., 2025b), allowing for parallelism, interpretability, and easier adaptation across domains. Given a question, the system first decomposes it into a sequence of subgoals. Each subgoal is then routed to a target KG selected based on schema compatibility and retrieval utility. Structured queries are generated under schema constraints and validated through a two-stage verification process. The resulting sub-answers are aggregated into a final response. The complete workflow is detailed in Algorithm 1.

Algorithm 1: AGENTICT^{2}S Pipeline for Heterogeneous KGQA

Require: Question q; KG collection $\mathcal{G} = \{G_1, \ldots, G_n\}$
Ensure: Final answer A

$\{g_1, \ldots, g_k\} \leftarrow \textsc{SubgoalParser}(q)$
for $i \leftarrow 1$ **to** k **do**
  $\mathcal{C}_i \leftarrow \textsc{CandidateGraphs}(g_i, \mathcal{G})$
  $G_i \leftarrow \textsc{Allocator}(g_i, \mathcal{C}_i)$ ▷ degenerates when $n = 1$
  $s_i \leftarrow \textsc{SPARQLSynthesizer}(g_i, G_i)$
  if $\textsc{Verifier}(s_i, G_i)$ **passes then**
    $A_i \leftarrow \textsc{ExecuteQuery}(s_i, G_i)$
  else
    $A_i \leftarrow \textsc{FallbackOrSkip}(g_i)$
  end if
end for
$\mathcal{R} \leftarrow \{(g_i, G_i, s_i, A_i)\}_{i=1}^{k}$
$A \leftarrow \textsc{ResultIntegrator}(\mathcal{R}, q)$
return A

4.1 Compositional Subgoal Parser Agent

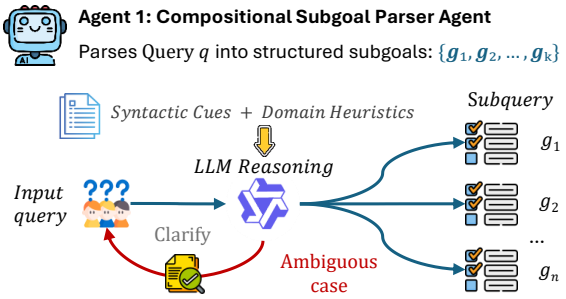


Figure 2: The overall workflow of the Compositional Subgoal Parser Agent.

The **Compositional Subgoal Parser Agent** initiates the AGENTICT^{2}S pipeline (Algorithm 1, line 1) by decomposing a natural language question into a sequence of subgoals, each corresponding to a basic reasoning operation such as entity retrieval, condition filtering, or aggregation (Figure 2). The parser employs a rule-based strategy that combines shallow syntactic analysis with domain-specific heuristics. Structured prompts are used to guide a language model in identifying syntactic patterns, segmenting the question into subgoals, and assigning intent labels based on linguistic structure. The agent also includes mechanisms to detect ambiguity or underspecified semantics; when triggered, these prompt the user for clarification to ensure subgoal completeness and correctness.

This approach does not require task-specific training data and is generalized across heterogeneous KGs. By decomposing complex queries into semantically independent components, the parser simplifies the reasoning process and enhances coordination among downstream agents.

4.2 Hierarchical Alignment Allocator Agent

Each subgoal is assigned to a relevant KG by the **Hierarchical Alignment Allocator Agent** (Algorithm 1, line 4), which selects candidate graphs through a two-stage retrieval process: coarse-grained filtering and fine-grained schema alignment based on the subgoal's intent and constraints (Figure 3).

In the **weak retrieval** stage, the allocator encodes each subgoal and each KG's schema summary (with domain labels) using `thenlper/gte-small`, and retrieves candidate KGs by cosine similarity in the embedding space. For each selected graph, a secondary search is conducted over its internal source files, such as ontology documents and schema descriptions, to locate relevant terms associated with the subgoal. In the **strong retrieval** stage, schema-level alignment is performed on the top-ranked candidates. This step extracts predicates, class hierarchies, and domain-range constraints to assess whether the graph's structure is compatible with the subgoal.

The KG pool and metadata index support incremental updates, allowing the system to incorporate new resources without reprocessing. As additional KGs and metadata become available, the allocator adjusts its retrieval strategy to maintain schema alignment. Retrieval decisions are further refined using utility scores from downstream verification, enabling the system to respond to schema drift and improve retrieval precision in complex queries.

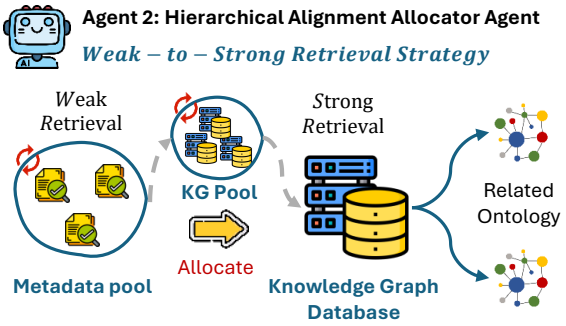


Figure 3: The overall workflow of the Hierarchical Alignment Allocator Agent.

4.3 Pattern-Driven SPARQL Synthesizer Agent

The **Pattern-Driven SPARQL Synthesizer Agent** (Algorithm 1, line 5) generates executable SPARQL queries from verified subgoals in three stages: query skeleton design, schema grounding, and post-hoc decoding (Figure 4).

In the **query skeleton design** stage, the agent selects a query template from a predefined library based on the subgoal's intent, such as entity retrieval, predicate filtering, or attribute matching. Each template represents an abstract SPARQL pattern and is parameterized with schema elements obtained from the previous agent. The template is instantiated with these elements to produce an initial query structure that is consistent with the schema and semantics of the target KG.

The **schema grounding** stage refines this structure by incorporating ontology-level constraints, including predicate types, class hierarchies, and domain-range specifications. This step replaces

generic placeholders with concrete schema components and enforces compatibility between entities, relations, and types defined in the graph. The resulting query is semantically coherent and structurally valid within the context of the selected KG.

To ensure reliability under diverse query conditions, the final stage, **post-hoc decoding**, serves as a model-agnostic validation layer that enforces structural and semantic correctness. Operating independently of the generation model, it systematically verifies syntax, resolves prefixes, and checks predicate usage, variable bindings, and schema alignment. In addition, it identifies query structures that are prone to execution failure, such as those that produce empty answer sets or violating ontology constraints. By applying targeted corrections only when necessary, post-hoc decoding provides a final safeguard, producing SPARQL queries that are not only valid and executable, but also resilient to schema variability across KGs.

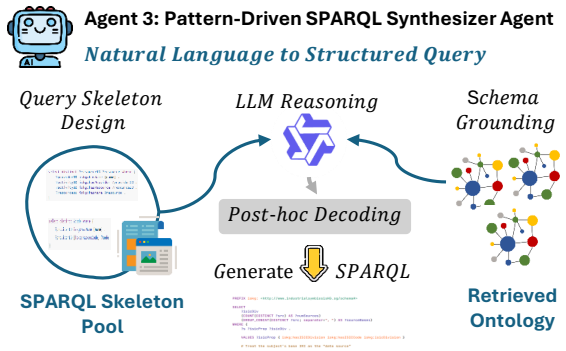


Figure 4: The overall workflow of the Pattern-Driven SPARQL Synthesizer Agent.

4.4 Dual-Stage Consistency Verifier Agent

To safeguard against execution errors and underspecified queries, the **Consistency Verifier Agent** (Algorithm 1, line 6) applies a two-stage verification process prior to query execution.

In the first stage, symbolic validation is applied to assess syntactic correctness, predicate usage, and type compatibility with the schema of the target KG. The agent also performs a preliminary execution check to ensure that the query yields a non-empty result under current graph conditions. The agent uses a bounded trial execution as a soft signal; empty results trigger revision or fallback.

In the second stage, counterfactual testing is used to evaluate the semantic specificity. The agent generates m=3 type-preserving perturbations (entity swaps, predicate swaps, and filter tighten/relax operations) and compares answer sets using Jaccard similarity. Queries with average similarity ≥ 0.8 across perturbations are flagged as overly general or underspecified. Only queries that pass both checks are forwarded for execution; others are revised or discarded. This mechanism improves the reliability of the final result by enforcing both structural validity and semantic precision.

4.5 Multi-Graph Consensus Aggregator Agent

Consensus Aggregator Agent (Algorithm 1, line 12) consolidates partial query results retrieved from multiple KGs. The agent first performs entity alignment across heterogeneous sources to resolve equivalence and redundancy. It then generates a unified natural language response by merging sub-answers using an LLM. This aggregation step ensures that the final output is consistent across sources and robust to variations in schema or terminology.

4.6 Complexity Analysis

For a query decomposed into k sub-goals, the end-to-end work of Algorithm 1 is $O(k)\big[\,|q|+n+\sum_{j=1}^{p}|\mathcal{S}_j|+BL+m\,T_{\text{exec}}+|G_j|\big]$, where $|q|$ is the question length, n the total number of KGs, $p \ll n$ the candidates that survive weak-tier retrieval, $\mathcal{S}_j$ the schema slice of graph G_j, B and L the beam width and decoded length in the synthesizer, m the counterfactual checks in the verifier, and T_{exec} the cost of one SPARQL evaluation on the executed graph of size $|G_j|$. All stages are embarrassingly parallel across sub-goals and graphs, so wall-clock latency is dominated by a handful of LLM calls and at most k verified query executions.

5 Experiments

5.1 Experimental Setup

We evaluate AGENTICT^{2}S and all baselines on a standardized text-to-SPARQL generation task, where each system receives a natural language question and generates a SPARQL query, which is executed against the relevant KGs. AGENTICT^{2}S runs in a hybrid setup that combines local model inference with API-based access to LLMs. The system is implemented using the LangGraph framework (LangGraph Contributors, 2024), which supports agent coordination, intermediate state tracking, and structured execution flow.

Local components include lightweight models for schema retrieval, subgoal parsing, and triple pattern scoring, executed on a workstation with four NVIDIA RTX 4090 GPUs. For language generation tasks that require larger context and structured decoding, we use `LLaMA3-70B` and `LLaMA3.3-70B` (Touvron et al., 2024), `Qwen3-32b` and `QwQ-32b` (Qwen Team, 2024) via Groq API (Groq, Inc., 2024). The remaining models are accessed via their official APIs. In the main experiments, all methods use `Qwen3-32b` as the shared backbone, additional comparisons across different backbone LLMs are reported in the Appendix C. All models are evaluated in zero-shot mode with a fixed temperature of 0.7 and no task-specific fine-tuning. Prompt and SPARQL query skeleton templates are provided in Appendix D and Appendix E, respectively.

5.2 Datasets and Evaluation Settings

We evaluate on three standard text-to-SPARQL benchmarks executed on Wikidata: LC-QuAD 2.0 (Dubey et al., 2019), QALD-9 Plus (Perevalov et al., 2022), and QALD-10 (Li et al., 2024). QALD-9 Plus extends QALD-9 by adding multilingual questions and transferring SPARQL queries from DBpedia to Wikidata; we use only the English split. QALD-10 increases query complexity, with 412 training questions derived from the QALD-9 Plus Wikidata split and a newly created test set of 394 questions reflecting real-world information needs.

In addition, we evaluate three in-house RDF knowledge graphs for industrial resource exchanges in the circular economy domain: **German-IS** (8,649 triples; 265 exchange cases), **EU-Pilot** (11,810 triples; 305 cases), and **Waste-Ledger** (52,402 triples; 1,727 waste/resource flows with quantitative attributes). We construct a benchmark of 328 natural-language queries spanning both single-KG and cross-KG settings over these graphs. Models are evaluated with three random seeds; we report mean ± standard deviation. Additional dataset details and benchmark construction are provided in Appendix A.1 and Appendix A.2.

5.3 Baselines

We compare against three categories of baselines:

Zero-Shot LLMs. Models prompted to generate SPARQL from natural language, without task-specific fine-tuning. This includes `LLaMA3-70B` and `LLaMA3.3-70B` (Touvron et al., 2024), `Qwen3-32b` and `QwQ-32b` (Qwen Team, 2024), `DeepSeek-V3` (Liu et al., 2024), and `GPT-4o` (OpenAI, 2024).

Reasoning-Augmented LLMs. The same models are evaluated with chain-of-thought prompting (Wei et al., 2022) and self-ask prompting (Press et al., 2023) to elicit intermediate reasoning steps prior to query generation.

Agent Controllers. General-purpose multi-agent frameworks, including ReAct (Yao et al., 2023) and AutoGen (Wu et al., 2024), are combined with LLaMA3 and Qwen3. These systems define planning and execution roles, but do not incorporate symbolic verification or schema-guided generation.

5.4 Evaluation Metrics

We evaluated model performance using four metrics: execution accuracy (EA), query syntax correctness (QSC), triple-level F1 (TF1), and average token usage (ATU). EA measures whether the generated SPARQL query executes without error and returns an answer set that matches the gold standard. QSC reflects the proportion of syntactically valid queries based on Apache Jena parsing. TF1 computes micro-averaged F1 over triple patterns in the WHERE clause, ignoring variable renaming and FILTER expressions. ATU reports the average number of tokens used per prediction as a proxy for generation efficiency. Detailed of these metrics are in Appendix A.3.

5.5 Main Results

Table 1 reports EA, QSC, and TF1 on four datasets, with all methods evaluated under a matched-backbone setting and results averaged over three random seeds (mean ± standard deviation). Overall, AGENTICT^{2}S achieves the best performance on every dataset and metric.

Performance across Wikidata benchmarks. On LC-QuAD 2.0, AGENTICT^{2}S reaches **84.13** ± **1.20** EA, **89.45** ± **1.11** QSC, and **87.08** ± **1.42** TF1. This is a large improvement over the strongest agent-controller baseline in Table 1 (AutoGen: 65.06 EA / 71.82 QSC / 69.10 TF1), corresponding to absolute gains of **+19.07** EA, **+17.63** QSC, and **+17.98** TF1.

On QALD-9 Plus, AGENTICT^{2}S achieves **86.08** ± **1.15** EA, **90.50** ± **1.26** QSC, and **88.34** ± **1.07** TF1, improving over AutoGen (67.33 / 73.18 / 70.21) by **+18.75** EA, **+17.32** QSC, and **+18.13** TF1.

Table 1: Performance comparison of different configurations on structured KGQA across four datasets. All methods share the same LLM backbone and are evaluated against matched baselines.

Dataset	Method	EA (%)	QSC (%)	TF1 (%)
LC-QUAD 2.0	Vanilla LLM	47.63 ± 1.20	58.00 ± 1.22	52.13 ± 1.14
	Reasoning LLM	54.02 ± 1.12	61.23 ± 1.07	58.26 ± 1.10
	ReAct Agent	63.77 ± 1.20	68.60 ± 1.45	65.73 ± 1.31
	AutoGen Agent	65.06 ± 1.17	71.82 ± 1.23	69.10 ± 1.44
	AGENTICT^{2}S (Ours)	**84.13 ± 1.20**	**89.45 ± 1.11**	**87.08 ± 1.42**
QALD-9 Plus	Vanilla LLM	56.20 ± 1.32	60.17 ± 1.56	58.21 ± 1.45
	Reasoning LLM	63.33 ± 1.24	66.02 ± 1.07	65.46 ± 1.28
	ReAct Agent	65.90 ± 1.11	69.62 ± 1.14	67.00 ± 1.28
	AutoGen Agent	67.33 ± 1.27	73.18 ± 1.47	70.21 ± 1.46
	AGENTICT^{2}S (Ours)	**86.08 ± 1.15**	**90.50 ± 1.26**	**88.34 ± 1.07**
QALD-10	Vanilla LLM	31.40 ± 1.18	37.00 ± 1.33	35.58 ± 1.06
	Reasoning LLM	36.27 ± 1.48	40.50 ± 1.13	38.31 ± 1.20
	ReAct Agent	40.13 ± 1.22	44.57 ± 1.26	42.71 ± 1.33
	AutoGen Agent	43.18 ± 1.27	46.52 ± 1.41	45.00 ± 1.34
	AGENTICT^{2}S (Ours)	**67.10 ± 1.17**	**71.55 ± 1.10**	**69.80 ± 1.30**
Circular Economy	Vanilla LLM	20.02 ± 1.03	24.73 ± 1.22	23.00 ± 1.00
	Reasoning LLM	40.25 ± 1.05	45.68 ± 1.82	42.72 ± 1.01
	ReAct Agent	47.13 ± 1.45	54.65 ± 2.01	52.30 ± 1.88
	AutoGen Agent	49.06 ± 1.12	58.39 ± 1.93	53.28 ± 1.62
	AGENTICT^{2}S (Ours)	**72.43 ± 1.24**	**76.21 ± 1.19**	**83.68 ± 0.39**

On QALD-10, which contains substantially more complex queries, AGENTICT^{2}S attains **67.10 ± 1.17** EA, **71.55 ± 1.10** QSC, and **69.80 ± 1.30** TF1, exceeding AutoGen (43.18 / 46.52 / 45.00) by **+23.92** EA, **+25.03** QSC, and **+24.80** TF1. The larger gains on QALD-10 indicate that explicit decomposition and validation become increasingly important as query structure and compositional constraints grow.

Performance on the Circular Economy benchmark. On the domain-specific Circular Economy benchmark (single-KG and cross-KG queries across three in-house RDF graphs), AGENTICT^{2}S achieves **72.43 ± 1.24** EA, **76.21 ± 1.19** QSC, and **83.68 ± 0.39** TF1. Compared to AutoGen (49.06 / 58.39 / 53.28), this yields absolute improvements of **+23.37** EA, **+17.73** QSC, and **+30.40** TF1. Notably, the particularly large TF1 gain suggests that the proposed framework more reliably grounds generation in the correct schema-level relations, which is critical for heterogeneous and schema-rich industrial KGs. We additionally report KGs allocation performance in Appendix B to characterize cross-KG routing behavior on cross-KG queries.

Comparison to prompting-based reasoning baselines. Reasoning-augmented prompting improves over vanilla prompting consistently, for example, on Circular Economy, EA increases from 20.02 to 40.25. However, AGENTICT^{2}S remains substantially ahead across all datasets. For example, on QALD-10, AGENTICT^{2}S improves over the Reasoning LLM baseline by **+30.83** EA and **+31.49** TF1; on Circular Economy, it improves by **+32.18** EA and **+40.96** TF1. These gaps indicate that prompting alone is insufficient to reliably satisfy structural constraints and KG-specific semantics, whereas modular reasoning combined with validation yields more robust SPARQL synthesis.

Aggregating results as an unweighted macro-average over the four datasets, AGENTICT^{2}S achieves 77.44 EA, 81.91 QSC, and 82.23 TF1, outperforming the strongest baseline in Table 1 (AutoGen) by **+21.28** EA points on average. Overall, these results show that the proposed agentic decomposition, schema grounding, and verification pipeline improves both syntactic validity and

Table 2: Ablation study on the contribution of key components in the AGENTICT^{2}S framework. Results reported on circular economy datasets.

Configuration	EA (%)	QSC (%)	TF1(%)
Ours w/o Decomposer Agent	23.68 ± 0.66	45.31 ± 0.19	45.42 ± 1.32
Ours w/o Allocator Agent	25.12 ± 0.18	31.02 ± 1.12	37.08 ± 0.36
Ours w/o Synthesizer Agent	–	–	–
Ours w/o Verifier Agent	55.12 ± 0.56	60.23 ± 1.10	66.17 ± 0.23
Ours w/o Agentic Collaborative	43.43 ± 0.78	52.56 ± 0.50	47.22 ± 0.16
Full AGENTICT^{2}S	$\mathbf{72.43 \pm 1.24}$	$\mathbf{76.21 \pm 1.19}$	$\mathbf{83.68 \pm 0.39}$

execution-level accuracy, with especially large benefits on complex and cross-KG query settings.

5.6 Ablation Study

To assess the contribution of each component, we perform ablations by disabling the **Decomposer Agent** (no subgoal parsing; pass q directly to the SPARQL generator), the **Allocator Agent** (replace graph selection with BM25 retrieval over the full KG pool), the **Synthesizer Agent** (skip query generation and return retrieved schema triples as pseudo-answers), and the **Verifier Agent** (execute queries without symbolic or counterfactual validation), and by removing **Agentic Collaboration** (execute all subtasks in a monolithic pipeline without scheduling).

As shown in Table 2, removing the subgoal decomposer leads to a sharp drop in EA from 72.4% to 23.7%, underscoring the importance of compositional parsing. Disabling the allocator similarly reduces performance, reflecting the impact of semantic drift when retrieval is not schema-aware. Omitting the verifier results in a 17% decline in EA, indicating that many syntactically valid queries are semantically underspecified. These results confirm the role of modular reasoning and symbolic validation in achieving accurate and robust SPARQL generation.

5.7 Qualitative Analysis

Figure 7 in Appendix F provides a step-by-step example of query processing under the AGENTICT^{2}S framework. The initial input is incomplete and underspecified: "For product code found in the resources, which trade codes co-occur with it?" The system initiates a clarification step, reformulating the input into a fully specified intent that references waste classification codes and requests cooccurrence information.

The clarified question is decomposed into subgoals, which are dispatched to the appropriate KGs (EU-Pilot and Waste-Ledger) based on schema compatibility. SPARQL queries are then generated for each subgoal using pattern-based templates. During verification, a syntactic and semantic check identifies a non-existent predicate in the initial output. The system modifies the query accordingly, replacing the faulty predicate with the correct one.

Following validation, verified queries are executed and their outputs aggregated. The final output includes both the SPARQL representation and a natural language statement indicating the mapping between CPA code `011150` and HS code `100610`. The example illustrates the role of clarification, decomposition, schema validation, and multi-graph answer fusion in the overall pipeline.

6 Conclusion

We presented AGENTICT^{2}S, a modular multi-agent framework for robust text-to-SPARQL generation over heterogeneous knowledge graphs. The framework decomposes questions into subgoals, allocates each subgoal via weak-to-strong schema alignment, synthesizes schema-grounded SPARQL using pattern-driven templates with post-hoc repair, and validates candidates using a dual-stage verifier combining symbolic and counterfactual consistency checks. Experiments on three Wikidata benchmarks and a domain-specific circular economy benchmark show consistent improvements over matched prompting and general-purpose agent controllers, with the largest gains on higher-complexity queries. Ablations further confirm that subgoal decomposition and verification are critical to execution-level accuracy. Future work will extend support for more advanced SPARQL constructs, improve robustness under schema drift, and optimize end-to-end latency and token cost.

Limitations

Our framework has several limitations. First, although AGENTICT^{2}S supports both heterogeneous multi-KG collections and the standard single-endpoint setting, its overall performance still depends on the availability and quality of schema metadata; weak or outdated ontology documentation can lead to mis-allocation and schema grounding errors. Second, the multi-agent pipeline introduces additional engineering complexity and runtime overhead compared to a single-shot text-to-SPARQL baseline, since it may require multiple LLM calls, verification steps, and (optionally) retries; the latency and cost trade-offs may be unfavorable in strict real-time scenarios. Third, our evaluation focuses on a limited set of KGs and query distributions; generalization to other domains, endpoints with different SPARQL dialect quirks, or highly complex queries (e.g., nested subqueries, advanced aggregates, or heavy use of OPTIONAL/UNION) may require extending the template library and verification rules.

Ethics Statement

Our experiments are conducted on knowledge graphs that are either publicly available or curated with appropriate authorization. The proposed framework does not require training on personal or sensitive data, and we do not intentionally collect, store, or disclose private information.

References

Sulaiman Olusegun Atiku. 2020. Knowledge Management for the Circular Economy. In *Handbook of Research on Entrepreneurship Development and Opportunities in Circular Economy*, pages 520–537. IGI Global.

Ricardo Barbosa, Ricardo Santos, and Paulo Novais. 2024. Collaborative Problem-solving with LLM: a Multi-agent System Approach to Solve Complex Tasks Using Autogen. In *International Conference on Practical Applications of Agents and Multi-Agent Systems*, pages 203–214. Springer.

Yuanning Cui, Zequn Sun, and Wei Hu. 2024. A Prompt-based Knowledge Graph Foundation Model for Universal In-Context Reasoning. *Advances in Neural Information Processing Systems*, 37:7095–7124.

Mohnish Dubey, Debayan Banerjee, Abdelrahman Abdelkawi, and Jens Lehmann. 2019. LC-QuAD 2.0: A Large Dataset for Complex Question Answering over Wikidata and DBpedia. In *International Semantic Web Conference*, pages 69–78. Springer.

Darren Edge, Ha Trinh, Newman Cheng, Joshua Bradley, Alex Chao, Apurva Mody, Steven Truitt, Dasha Metropolitansky, Robert Osazuwa Ness, and Jonathan Larson. 2024. From Local to Global: A Graph RAG Approach to Query-focused Summarization. *arXiv preprint arXiv:2404.16130*.

Yunfan Gao, Yun Xiong, Xinyu Gao, Kangxiang Jia, Jinliu Pan, Yuxi Bi, Yixin Dai, Jiawei Sun, Haofen Wang, and Haofen Wang. 2023. Retrieval-augmented Generation for Large Language Models: A Survey. *arXiv preprint arXiv:2312.10997*, 2(1).

Groq, Inc. 2024. Groq api and lpu inference platform. https://groq.com. Accessed: 2024-07-29.

Taicheng Guo, Xiuying Chen, Yaqi Wang, Ruidi Chang, Shichao Pei, Nitesh V Chawla, Olaf Wiest, and Xiangliang Zhang. 2024. Large Language Model Based Multi-agents: A Survey of Progress and Challenges. In *33rd International Joint Conference on Artificial Intelligence (IJCAI 2024)*.

Haoyu Han, Yu Wang, Harry Shomer, Kai Guo, Jiayuan Ding, Yongjia Lei, Mahantesh Halappanavar, Ryan A Rossi, Subhabrata Mukherjee, Xianfeng Tang, and 1 others. 2024. Retrieval-augmented Generation with Graphs (GraphRAG). *arXiv preprint arXiv:2501.00309*.

Bolei He, Nuo Chen, Xinran He, Lingyong Yan, Zhenkai Wei, Jinchang Luo, and Zhen-Hua Ling. 2024. Retrieving, Rethinking and Revising: The Chain-of-Verification Can Improve Retrieval Augmented Generation. In *Findings of the Association for Computational Linguistics: EMNLP 2024*, pages 10371–10393.

Sirui Hong, Mingchen Zhuge, Jonathan Chen, Xiawu Zheng, Yuheng Cheng, Jinlin Wang, Ceyao Zhang, Zili Wang, Steven Ka Shing Yau, Zijuan Lin, and 1 others. 2023. MetaGPT: Meta programming for a multi-agent collaborative framework. In *The Twelfth International Conference on Learning Representations*.

Yixin Ji, Kaixin Wu, Juntao Li, Wei Chen, Mingjie Zhong, Xu Jia, and Min Zhang. 2024. Retrieval and Reasoning on KGs: Integrate Knowledge Graphs into Large Language Models for Complex Question Answering. In *Findings of the Association for Computational Linguistics: EMNLP 2024*, pages 7598–7610.

Ziwei Ji, Nayeon Lee, Rita Frieske, Tiezheng Yu, Dan Su, Yan Xu, Etsuko Ishii, Ye Jin Bang, Andrea Madotto, and Pascale Fung. 2023. Survey of Hallucination in Natural Language Generation. *ACM Computing Surveys*, 55(12):1–38.

LangGraph Contributors. 2024. Langgraph: Multi-agent workflow framework for llms. https://www.langgraph.dev. Accessed: 2024-07-29.

Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, and 1 others. 2020. Retrieval-augmented Generation for Knowledge-intensive NLP Tasks. *Advances in Neural Information Processing Systems*, 33:9459–9474.

Guohao Li, Hasan Hammoud, Hani Itani, Dmitrii Khizbullin, and Bernard Ghanem. 2023. CAMEL: Communicative Agents for "Mind" Exploration of Large Language Model Society. *Advances in Neural Information Processing Systems*, 36:51991–52008.

Yihao Li, Ru Zhang, and Jianyi Liu. 2024. An enhanced prompt-based llm reasoning scheme via knowledge graph-integrated collaboration. In *Artificial Neural Networks and Machine Learning – ICANN 2024*, pages 251–265, Cham. Springer Nature Switzerland.

Aixin Liu, Bei Feng, Bing Xue, Bingxuan Wang, Bochao Wu, Chengda Lu, Chenggang Zhao, Chengqi Deng, Chenyu Zhang, Chong Ruan, and 1 others. 2024. Deepseek-v3 technical report. *arXiv preprint arXiv:2412.19437*.

Lars-Peter Meyer, Johannes Frey, Felix Brei, and Natanael Arndt. 2024. Assessing SPARQL capabilities of large language models. *arXiv preprint arXiv:2409.05925*.

Bonan Min, Hayley Ross, Elior Sulem, Amir Pouran Ben Veyseh, Thien Huu Nguyen, Oscar Sainz, Eneko Agirre, Ilana Heintz, and Dan Roth. 2023. Recent Advances in Natural Language Processing via Large Pre-trained Language Models: A Survey. *ACM Computing Surveys*, 56(2):1–40.

OpenAI. 2024. GPT-4o Technical Report. https://openai.com/index/gpt-4o. OpenAI Technical Report.

Aleksandr Perevalov, Dennis Diefenbach, Ricardo Usbeck, and Andreas Both. 2022. QALD-9-plus: A Multilingual Dataset for Question Answering over DBpedia and Wikidata Translated by Native Speakers. In *2022 IEEE 16th International Conference on Semantic Computing (ICSC)*, pages 229–234.

Ofir Press, Muru Zhang, Sewon Min, Ludwig Schmidt, Noah A Smith, and Mike Lewis. 2023. Measuring and Narrowing the Compositionality Gap in Language Models. In *Findings of the Association for Computational Linguistics: EMNLP 2023*, pages 5687–5711.

Qwen Team. 2024. Qwen3: A Family of Open-Source Language Models by Alibaba Cloud. https://github.com/QwenLM/Qwen. Alibaba Cloud, Technical Report.

Apoorv Saxena, Aditay Tripathi, and Partha Talukdar. 2020. Improving Multi-Hop Question Answering over Knowledge Graphs Using Knowledge Base Embeddings. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 4498–4507.

Hugo Touvron and 1 others. 2024. LLaMA 3: Open Foundation and Instruction Models. https://ai.meta.com/blog/meta-llama-3. Meta AI.

Khanh-Tung Tran, Dung Dao, Minh-Duong Nguyen, Quoc-Viet Pham, Barry O'Sullivan, and Hoang D Nguyen. 2025. Multi-agent Collaboration Mechanisms: A Survey of LLMs. *arXiv preprint arXiv:2501.06322*.

W3C. 2013. SPARQL 1.1 Query Language. https://www.w3.org/TR/sparql11-query/. W3C Recommendation.

Fei Wang, Xingchen Wan, Ruoxi Sun, Jiefeng Chen, and Sercan O Arik. 2025. Astute RAG: Overcoming Imperfect Retrieval Augmentation and Knowledge Conflicts for Large Language Models. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 30553–30571, Vienna, Austria. Association for Computational Linguistics.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed Chi, Quoc V Le, Denny Zhou, and 1 others. 2022. Chain-of-Thought Prompting Elicits Reasoning in Large Language Models . *Advances in Neural Information Processing Systems*, 35:24824–24837.

Junde Wu, Jiayuan Zhu, Yunli Qi, Jingkun Chen, Min Xu, Filippo Menolascina, Yueming Jin, and Vicente Grau. 2025a. Medical Graph RAG: Evidence-based Medical Large Language Model via Graph Retrieval-augmented Generation. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 28443–28467.

Qingyun Wu, Gagan Bansal, Jieyu Zhang, Yiran Wu, Beibin Li, Erkang Zhu, Li Jiang, Xiaoyun Zhang, Shaokun Zhang, Jiale Liu, and 1 others. 2024. AutoGen: Enabling Next-Gen LLM Applications via Multi-Agent Conversations. In *First Conference on Language Modeling*.

Ruofan Wu, Youngwon Lee, Fan Shu, Danmei Xu, Seung-won Hwang, Zhewei Yao, Yuxiong He, and Feng Yan. 2025b. ComposeRAG: A Modular and Composable RAG for Corpus-Grounded Multi-Hop Question Answering. *arXiv preprint arXiv:2506.00232*.

Zhao Xinjie, Fan Gao, Xingyu Song, Yingjian Chen, Rui Yang, Yanran Fu, Yuyang Wang, Yusuke Iwasawa, Yutaka Matsuo, and Irene Li. 2025. ReAgent: Reversible Multi-Agent Reasoning for Knowledge-Enhanced Multi-Hop QA. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pages 4067–4089.

Shicheng Xu, Liang Pang, Huawei Shen, Xueqi Cheng, and Tat-Seng Chua. 2024. Search-in-the-Chain: Interactively Enhancing Large Language Models with Search for Knowledge-intensive Tasks. In *Proceedings of the ACM Web Conference 2024*, pages 1362–1373.

Yingxuan Yang, Huacan Chai, Shuai Shao, Yuanyi Song, Siyuan Qi, Renting Rui, and Weinan Zhang. 2025. Agentnet: Decentralized Evolutionary Coordination for LLM-based Multi-agent Systems. *arXiv preprint arXiv:2504.00587*.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. 2023. ReAct: Synergizing Reasoning and Acting in Language Models. In *International Conference on Learning Representations (ICLR)*.

Michihiro Yasunaga, Hongyu Ren, Antoine Bosselut, Percy Liang, and Jure Leskovec. 2021. QA-GNN: Reasoning with Language Models and Knowledge Graphs for Question Answering. In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 535–546.

Appendix Contents

A Datasets and Evaluation Settings

A.1 Wikidata-based Text-to-SPARQL Benchmarks

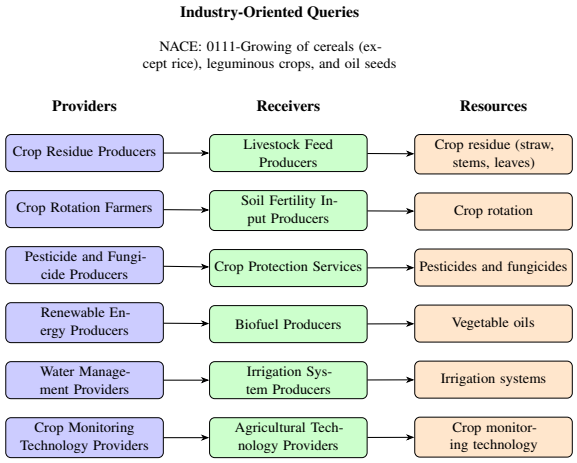
We evaluate on three widely used text-to-SPARQL benchmarks executed on Wikidata: LC-QuAD 2.0 (Dubey et al., 2019), QALD-9 Plus (Perevalov et al., 2022), and QALD-10 (Li et al., 2024). All datasets are evaluated using the same execution-based protocol (EA/QSC/TF1), and we follow the official train/test splits provided by each benchmark.

QALD-9 Plus extends QALD-9 by adding multilingual questions and transferring SPARQL queries from DBpedia to Wikidata; we use only the English split (371 train / 136 test).

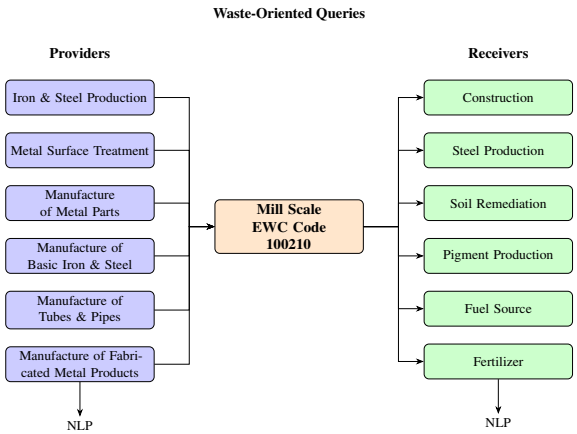
QALD-10 increases query complexity and realism. It provides 412 training questions derived from the QALD-9 Plus Wikidata split and introduces a new test set of 394 questions that reflect real-world information needs.

A.2 Circular Economy Knowledge Graphs

To assess robustness beyond Wikidata-style KGs, we evaluate on three RDF knowledge graphs curated in-house for industrial resource exchanges in the circular economy domain. These graphs differ in schema complexity, entity coverage, and the prevalence of quantitative attributes:



(a) Industry-Oriented Queries (NACE: 0111).



(b) Waste-Oriented Queries (EWC: 100210).

Figure 5: Examples of industry-oriented and waste-oriented query templates in circular economy KGs.

- **German-IS.** This dataset (8,649 triples) models 265 exchange cases from a national German program. Entities include `Provider`, `Receiver`, and `Resource`, with textual descriptions and annotations using WZ-08 (German Classification of Economic Activities), CPA (EU Product Classification), EWC (European Waste Catalog), ISIC (International Standard Industrial Classification), HS (Harmonized System Code) and SSIC (Singapore Standard Industrial Classification).

- **EU-Pilot.** This dataset (11,810 triples) covers 305 cases from an EU-funded project. It includes structured metadata such as `hasStatus` and `hasComments`, and is annotated with NACE Rev. 2, CPA and EWC codes.

- **Waste-Ledger.** This dataset (52,402 triples) contains 1,727 instances of waste and resource

flows across Europe. It includes quantitative attributes such as `hasGwp100` (global warming potential), and regulatory metadata such as `hasHSCode`, and `hasEWCCode`.

Figure 5 shows two representative query patterns supported by these KGs: industry-oriented queries that map `Provider–Receiver–Resource` relations under a target NACE code (Figure 5a), and waste-oriented queries centered on an EWC code that connect upstream producers to downstream receivers (Figure 5b).

A.3 Evaluation Metric Details

- **Execution Accuracy (EA).** A prediction is considered correct if (i) the generated query compiles and (ii) the resulting set of answers exactly matches the gold answer set. Formally,

$$\text{EA} = \frac{1}{N}\sum_{i=1}^{N} \mathbb{1}\big[\texttt{compile}(q_i) \wedge \texttt{ans}(q_i) = \texttt{ans}(q_i^{\text{gold}})\big] \quad (1)$$

  where q_i is the predicted query, q_i^{gold} is the gold query, and $\texttt{ans}(q)$ denotes the execution result of query q.

- **Query Syntax Correctness (QSC).** The proportion of generated queries that compile successfully using Apache Jena:

$$\text{QSC} = \frac{1}{N}\sum_{i=1}^{N} \mathbb{1}\left[\texttt{compile}(q_i)\right] \quad (2)$$

- **Triple-Level F1 (TF1).** This metric compares triple patterns in the WHERE clause of predicted and gold queries, ignoring variable renaming and FILTERs. Precision, recall, and F1 are computed with micro-averaging:

$$\text{TF1} = \frac{2 \cdot P \cdot R}{P + R} \quad (3)$$

$$P = \frac{\sum_{i=1}^{N} |\texttt{TP}_i|}{\sum_{i=1}^{N} |\texttt{Pred}_i|} \quad (4)$$

$$R = \frac{\sum_{i=1}^{N} |\texttt{TP}_i|}{\sum_{i=1}^{N} |\texttt{Gold}_i|} \quad (5)$$

  where $\texttt{TP}_i$ is the number of correct triple patterns for example i, $\texttt{Pred}_i$ is the number of predicted triple patterns, and $\texttt{Gold}_i$ is the number of gold triple patterns.

- **Average Token Usage (ATU).** The average number of input tokens per prediction, reflecting prompt-level cost:

$$\text{ATU} = \frac{1}{N}\sum_{i=1}^{N} \texttt{tokens}_{\text{in}}(q_i) \quad (6)$$

  where $\texttt{tokens}_{\text{in}}(q_i)$ denotes the number of input tokens used to prompt query q_i.

Note. N denotes the total number of evaluation samples. $\mathbb{1}[\cdot]$ is the indicator function that returns 1 when the condition is true and 0 otherwise.

B Knowledge Graph Allocation Performance

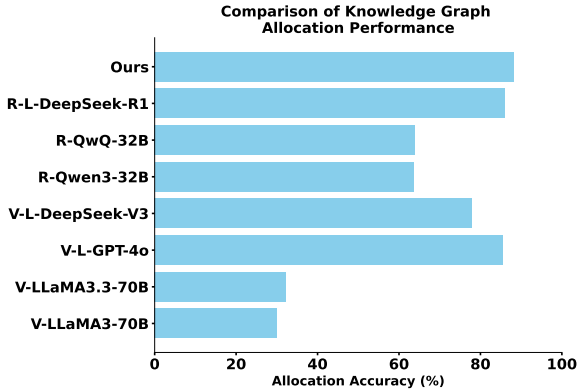


Figure 6: Comparison of knowledge graph allocation performance across models.

We evaluate the allocator's effectiveness by measuring the proportion of subgoals routed to a graph that actually contains the gold answer. As shown in Figure 6, AGENTICT^{2}S attains the highest allocation accuracy, exceeding the strongest baseline (DeepSeek-R1) by over 10%. This precision directly impacts EA: correctly routed subgoals are more likely to match relevant schemas, leading to higher query validity and fewer empty results.

C Baseline LLM Performance Comparison

We benchmark diverse LLM backbones, including vanilla and reasoning-augmented variants, on circular economy KGs using EA, QSC, and TF1. Tables 3 and 4 show per-graph performance on EA and QSC respectively. Table 5 presents an overall comparison including token efficiency, demonstrating that our agentic approach achieves superior performance while maintaining reasonable computational costs.

Table 3: EA on single-KG.

Model	German-IS	EU-Pilot	Waste-Ledger
V-LLaMA3-70B	15.08 ± 1.86	16.05 ± 1.19	13.82 ± 2.75
V-LLaMA3.3-70B	17.21 ± 1.43	14.31 ± 0.80	14.28 ± 0.76
V-L-GPT-4o	76.50 ± 2.93	74.24 ± 1.03	72.05 ± 0.82
V-L-DeepSeek-V3	69.76 ± 2.63	68.22 ± 1.54	63.56 ± 2.36
R-Qwen3-32B	53.12 ± 2.64	54.55 ± 2.08	49.67 ± 0.67
R-QwQ-32B	56.09 ± 2.88	53.73 ± 1.43	51.82 ± 2.03
R-L-DeepSeek-R1	79.63 ± 2.67	74.86 ± 1.52	72.91 ± 0.94
Ours: AGENTICT^{2}S	**85.64 ± 1.06**	**87.72 ± 1.28**	**85.01 ± 2.33**

Table 4: QSC on single-KG.

Model	German-IS	EU-Pilot	Waste-Ledger
V-LLaMA3-70B	21.31 ± 0.75	25.12 ± 0.97	25.83 ± 0.60
V-LLaMA3.3-70B	27.48 ± 2.00	24.44 ± 1.72	24.35 ± 1.55
V-L-GPT-4o	86.55 ± 1.24	88.92 ± 1.07	86.44 ± 1.57
V-L-DeepSeek-V3	61.39 ± 1.80	67.23 ± 0.58	62.64 ± 1.90
R-Qwen3-32B	75.44 ± 1.28	73.64 ± 1.00	75.86 ± 1.53
R-QwQ-32B	67.49 ± 1.12	72.98 ± 0.84	68.28 ± 0.68
R-L-DeepSeek-R1	86.05 ± 0.81	87.75 ± 1.46	85.22 ± 1.89
Ours: AGENTICT^{2}S	**97.14 ± 1.62**	**96.99 ± 0.55**	**97.91 ± 1.44**

Table 5: Overall performance comparison of different model configurations on structured KGQA. AGENTICT^{2}S achieves the highest accuracy across all metrics while maintaining lower token usage.

Model	EA (%)	QSC (%)	TF1 (%)	ATU
V-LLaMA3-70B	10.12 ± 2.45	15.47 ± 3.12	18.36 ± 2.78	
V-LLaMA3.3-70B	12.28 ± 2.13	18.65 ± 3.08	10.52 ± 2.33	1632
V-L-GPT-4o	55.15 ± 1.31	60.04 ± 0.56	58.32 ± 1.12	(± 110)
V-L-DeepSeek-V3	45.74 ± 1.22	50.19 ± 1.41	47.09 ± 1.13	
R-Qwen3-32B	40.25 ± 1.05	45.68 ± 1.82	42.72 ± 1.01	1632
R-QwQ-32B	35.67 ± 1.01	40.31 ± 0.75	38.25 ± 0.89	
R-L-DeepSeek-R1	50.32 ± 1.14	55.21 ± 1.22	52.18 ± 0.07	(± 110)
A-ReAct+LLaMA3-70B	28.52 ± 1.03	37.33 ± 1.74	33.25 ± 0.98	
A-ReAct+Qwen3-32B	47.13 ± 1.45	54.65 ± 2.01	52.30 ± 1.88	1518
A-AutoGen+LLaMA3-70B	31.21 ± 2.17	33.02 ± 0.89	37.14 ± 1.06	(± 110)
A-AutoGen+Qwen3-32B	49.06 ± 1.12	58.39 ± 1.93	53.28 ± 1.62	
Ours: AGENTICT^{2}S	**72.43 ± 1.24**	**76.21 ± 1.19**	**83.68 ± 0.39**	**875 ± 90**

Note: V = Vanilla LLM, R = Reasoning LLM, L = Large-Scale Pretrained Model, A = Agent Framework.

D Prompt Template

SPARQL Generation Prompt for LLM Testing

Instruction:
Answer the question below by generating a query that uses the structure of the knowledge graph. You are provided with information about the relevant types and relationships in the graph. Do not include explanation, only generate the query that would retrieve the desired result.

Question:
What are the top 10 most frequently used EWC codes based on the number of associated exchange records in the `ressource-deutschland.de` knowledge graph?

Knowledge Graph Information:

- Each exchange is an entity of type `iskg:Database_ID`.
- ...

[The prompt provides only basic schema-level information from the knowledge graph, with no result-oriented guidance or implicit task hints.]

Graph Allocation Prompt for LLM Testing

Instruction: This is a knowledge graph allocation task. Based on the provided schema-level background information, select one or more appropriate named graphs to answer the question. Do not include explanations, only output the name(s) of the knowledge graph(s).

Question:
Identify the HS codes that co-occur with each CPA code. Determine which named graph(s) should be queried, and group the output by graph.

Knowledge Graph Information:

- **Graph 1 (German-IS):** Background schema information ...
- **Graph 2 (EU-Pilot):** Background schema information ...
- **Graph 3 (Waste-Ledger):** Background schema information ...
- ...

[The prompt provides only basic schema-level information from the knowledge graph, with no result-oriented guidance or implicit task hints.]

Note. This prompt is intended for illustrative purposes only. In actual LLM evaluations, the template should be supplemented with concrete schema details, representative examples, namespace declarations, and any other relevant contextual information required to support realistic and fair testing.

E SPARQL Query Skeleton Templates

Provider-based Queries

1. Search provider by code

```
SELECT DISTINCT ?providerID ?provider WHERE {
    ?providerID iskg:hasxxxCode [code] .
    ?providerID iskg:hasName ?provider .
}
```

2. Search provider code by name

```
SELECT DISTINCT ?code WHERE {
    ?providerID iskg:hasName [name] .
    ?providerID iskg:hasxxxCode ?code .
}
```

Receiver-based Queries

3. Search receiver by code

```
SELECT DISTINCT ?receiverID ?receiver WHERE {
    ?receiverID iskg:hasxxxCode [code] .
    ?receiverID iskg:hasName ?receiver .
}
```

4. Search receiver code by name

```
SELECT DISTINCT ?code WHERE {
    ?receiverID iskg:hasName [name] .
    ?receiverID iskg:hasxxxCode ?code .
}
```

Resource-based Queries

5. Search resource by code

```
SELECT DISTINCT ?resourceID ?resource WHERE {
    ?resourceID iskg:hasxxxCode [code] .
    ?resourceID iskg:hasName ?resource .
}
```

6. Search resource code by name

```
SELECT DISTINCT ?code WHERE {
    ?resourceID iskg:hasName [name] .
    ?resourceID iskg:hasxxxCode ?code .
}
```

7. Search resource by provider name

```
SELECT DISTINCT ?resourceID ?resource WHERE {
    ?providerID iskg:hasName [name] .
    ?activityID iskg:hasProvider ?providerID .
    ?activityID iskg:hasResource ?resourceID .
    ?resourceID iskg:hasName ?resource .
}
```

8. Search resource by receiver name

```
SELECT DISTINCT ?resourceID ?resource WHERE {
    ?receiverID iskg:hasName [name] .
    ?activityID iskg:hasReceiver ?receiverID .
    ?activityID iskg:hasResource ?resourceID .
    ?resourceID iskg:hasName ?resource .
}
```

Note. These SPARQL templates are manually constructed by domain experts and organized in a schema-driven yet dataset-agnostic manner: we derive common query intents (entity lookup by name/code, relation traversal, and basic filtering) and encode them as SPARQL 1.1 skeletons with slots such as `[name]` and `[code]`. KG-specific predicates are abstracted as placeholders (e.g., `iskg:hasxxxCode`) and instantiated via a lightweight per-dataset mapping layer; in our circular economy KGs this mapping corresponds to `Provider/Receiver/Resource` roles and the exchange node (e.g., `?activityID`).

F Case Study

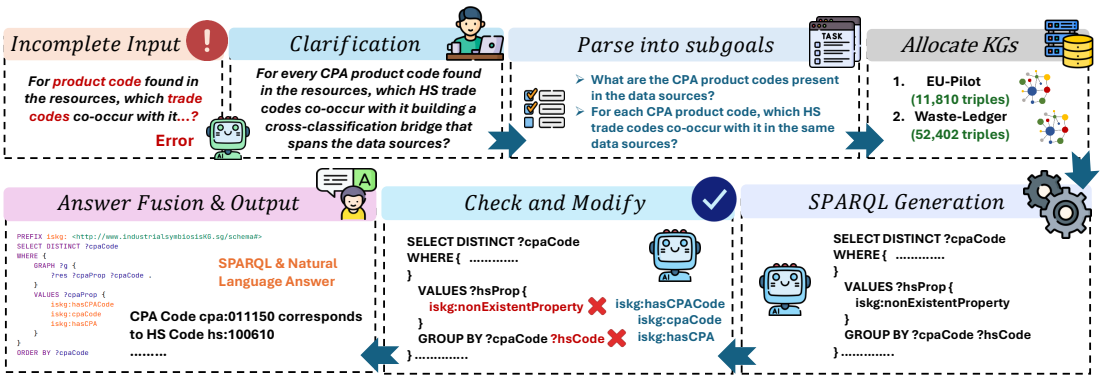


Figure 7: Example workflow of AGENTICT^{2}S applied to a circular economy query.

Case 1: Single KG

Question: Count of industrial symbioses grouped by provider NACE classification

Target KG: German-IS

LLaMA3-70B ✗

```
PREFIX iskg: <https://iskgontology.org/>
SELECT ?naceCode (COUNT(?exchange) AS ?symbiosisCount)
WHERE {
  ?exchangeURI <https://maestri-spire.eu/> .
  ?exchangeURI iskg:hasProvider ?provider .
  ?provider iskg:hasNACECode ?naceCode .
}
GROUP BY ?naceCode
GROUP BY ?country
```

Error: ?exchangeURI <https://maestri-spire.eu/ >

This line attempts to use `<https://maestri-spire.eu/>` as a predicate (property) to connect `?exchangeURI`, but this is a URL rather than a valid RDF predicate, and its semantics are unclear.

Qwen3-32B ✓

```
PREFIX iskg: <http://www.industrialsymbiosisKG.sg/schema#>
SELECT ?nace (COUNT(?exchange) AS ?symbiosesCount)
WHERE {
  ?exchange iskg:hasProvider ?provider .
  ?provider iskg:hasNACECode ?nace .
  FILTER (STRSTARTS(STR(?exchange), "https://maestri-spire.eu"))
}
GROUP BY ?nace
```

Ours: AgenticT^{2}S ✓

```
PREFIX iskg: <http://www.industrialsymbiosisKG.sg/schema#>

SELECT ?naceCode (COUNT(?exchange) AS ?symbiosisCount)
WHERE {
  ?exchange iskg:hasProvider ?provider .
  ?provider iskg:hasNACECode ?naceCode .
  FILTER(STRSTARTS(STR(?exchange), "https://maestri-spire.eu"))
}
GROUP BY ?naceCode
```

Case 2: Single KG

Question: Temporal distribution of exchanges by decade of publication

Target KG: EU-Pilot

LLaMA3-70B ✗

```
PREFIX iskg: <https://www.ressource-deutschland.de/iskg/>
PREFIX res: <https://www.ressource-deutschland.de/>
SELECT (floor(?year / 10) * 10) as ?decade (COUNT(*) as ?count)
WHERE {
  ?exchange iskg:hasInformation ?info .
  ?info iskg:hasPublishedYear ?year .
  FILTER(STRSTARTS(STR(?exchange),
  "https://www.ressource-deutschland.de"))
}
GROUP BY (floor(?year / 10) * 10)
ORDER BY ?decade
```

Error: SELECT (floor(?year / 10) * 10) AS ?decade (COUNT(*) AS ?count) GROUP BY (floor(?year / 10) * 10)

Expressions like `floor(?year / 10) * 10` cannot be used directly in `SELECT` or `GROUP BY`; use `BIND(... AS ?var)` instead.

Qwen3-32B ✗

```
SELECT ?decade (COUNT(*) AS ?count) WHERE {
  ?exchange a iskg:Database_ID ;
            iskg:hasInformation ?info .
  ?info iskg:hasPublishedYear ?year .
  FILTER(STR(?exchange) STARTS WITH
  "https://www.ressource-deutschland.de")
  BIND(FLOOR(xsd:integer(?year) / 10) * 10 AS ?decade)
} GROUP BY ?decade ORDER BY ?decade
```

Error: `STARTS WITH` is not valid SPARQL syntax; use `STRSTARTS(string, prefix)` instead.

Ours: AgenticT^{2}S ✓

```
PREFIX iskg: <https://www.ressource-deutschland.de/iskg/>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
SELECT ?decade (COUNT(*) AS ?count)
WHERE {
  ?exchange a iskg:Database_ID ;
            iskg:hasInformation ?info .
  ?info iskg:hasPublishedYear ?year .
  FILTER(STRSTARTS(STR(?exchange),
  "https://www.ressource-deutschland.de"))
  BIND(FLOOR(xsd:integer(?year) / 10) * 10 AS ?decade)
}
GROUP BY ?decade
ORDER BY ?decade
```

Case 3: Cross KG

Question: Mapping co-occurrences between CPA product codes and HS trade codes across data sources.

Target KG: German-IS & Waste-Ledger

LLaMA3-70B ✗

```
PREFIX iskg: <http://www.industrialsymbiosisKG.sg/schema#>
SELECT ?cpaCode (GROUP_CONCAT(DISTINCT STR(?hs); SEPARATOR=",")
AS ?hsCodes)
WHERE {
  GRAPH ?g {
    ?res iskg:hasCPACode ?cpaCode ;
         iskg:hasHSCode ?hs .
  }
GROUP BY ?cpaCode
```

Error: This is an incomplete output generated, where the model begins listing CPA–HS code co-occurrences but terminates prematurely without completing the final sentence or result set.

Qwen3-32B ✗

```
SELECT ?cpaCode (GROUP_CONCAT(DISTINCT ?hsCode; SEPARATOR=",")
AS ?hsList)
WHERE {
  ?entity <hasCPA> ?cpaCode .
  ?entity <hasHS> ?hsCode .
}
GROUP BY ?cpaCode
```

Error: This output exhibits typical issues such as the use of placeholder predicates (`<hasCPA>`, `<hasHS>`) instead of ontology-defined URIs, and a malformed `GROUP_CONCAT` syntax missing parentheses and proper separator formatting.

Ours: AgenticT^{2}S ✓

```
SELECT ?cpaCode (GROUP_CONCAT(DISTINCT STR(?hs); SEPARATOR=",")
AS ?hsCodes)
WHERE {
  GRAPH ?g {
    ?res iskg:hasCPACode ?cpaCode .
    FILTER EXISTS {
      ?res iskg:hasHSCode ?hs .
    }
  }
}
GROUP BY ?cpaCode
```

Merged version

G Notation Table

Table 6: Notations.

Symbol	Meaning
q	input natural-language question
$\mathcal{G} = \{G_1, \ldots, G_n\}$	collection of n knowledge graphs
G_i	the i-th knowledge graph (or selected graph for subgoal g_i in Alg. 1)
Ent	union of entity sets across graphs in $\mathcal{G}$
k	number of subgoals after decomposition
g_i	i-th subgoal derived from q
$\mathcal{C}_i$	candidate graph set for subgoal g_i
s_i	SPARQL query generated for subgoal g_i
A_i	answer set returned by executing s_i
A	final integrated answer produced by the result integrator
p	number of candidate graphs kept after weak-tier retrieval ($p \ll n$)
$\mathcal{S}_j$	schema slice (predicates/types/constraints) extracted from candidate graph G_j
B, L	beam width and decoded length in the synthesizer
m	number of counterfactual checks in the verifier
T_{exec}	cost of one SPARQL execution on the target endpoint/graph