BrickQA: Bridging the Semantic Gap in Building Operations with Dynamic Graph Exploration

Yun-Dam Ko
Stanford University
Stanford, USA
yundamko@stanford.edu

Hyeyun Eunice Jung
Stanford University
Stanford, USA
ehjung@stanford.edu

Marco Pritoni
Lawrence Berkeley National Laboratory
Berkeley, USA
mpritoni@lbl.gov

Rishee Jain
Stanford University
Stanford, USA
rishee.jain@stanford.edu

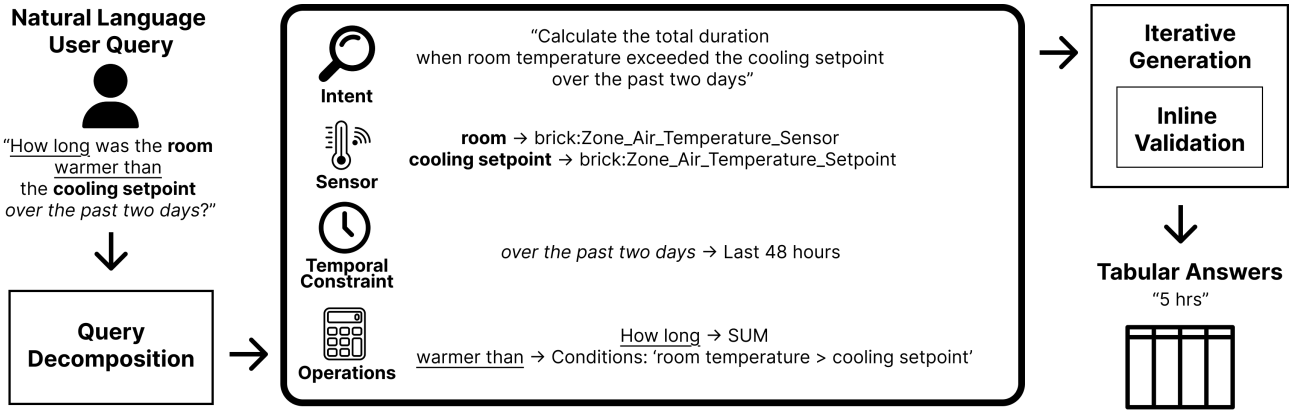

Figure 1: Overview of BrickQA

Abstract

While standardized ontologies like the Brick schema address data heterogeneity in Building Automation Systems (BAS), accessing this semantic data remains a challenge as domain experts often lack the expertise to formulate complex SPARQL queries. To bridge this gap, we present **BrickQA**, a Large Language Model (LLM)-based framework that translates natural language into executable SPARQL queries through structured query decomposition, dynamic schema exploration, and inline validation. BrickQA utilizes an iterative reasoning agent to actively navigate graph topology through dynamic exploration actions without requiring exhaustive context injection or model fine-tuning. This approach effectively mitigates hallucinations, particularly in large-scale building knowledge graphs. Empirical evaluation on BuildingQA, a standardized benchmark, demonstrates that BrickQA significantly outperforms ReAct baselines, delivering a 0.291–0.355 absolute F1 improvement while achieving 3×–12.7× higher token cost-efficiency. Beyond these metrics, the framework maintains structural fidelity across heterogeneous buildings and remains resilient to ambiguous queries without requiring site-specific fine-tuning. Furthermore, a case study on operational analytics validates the framework's capability to handle temporal and aggregation constraints, effectively transforming abstract semantic models into actionable facility management insights.[1]

CCS Concepts

• **Information systems** → **Query languages**; • **Computing methodologies** → **Artificial intelligence**.

Keywords

Large Language Models, Knowledge Graphs, Knowledge Graph Question Answering, Building Ontologies

ACM Reference Format:
Yun-Dam Ko, Hyeyun Eunice Jung, Marco Pritoni, and Rishee Jain. 2026. BrickQA: Bridging the Semantic Gap in Building Operations with Dynamic Graph Exploration. In *The 13th ACM International Conference on Systems for Energy-Efficient Buildings, Cities, and Transportation (BuildSys '26), June 22–25, 2026, Banff, AB, Canada.* ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/3744256.3812570

BuildSys '26, Banff, AB, Canada

ACM ISBN 979-8-4007-2012-3/26/06
https://doi.org/10.1145/3744256.3812570

[1] Code available at https://github.com/yun-dam/BrickQA

1 Introduction

Building Automation Systems (BAS) collect massive datasets from subsystems such as Heating, Ventilation, and Air Conditioning (HVAC), yet the lack of semantic standardization hinders their potential [5]. Inconsistent, vendor-specific metadata obstructs scalable applications such as intelligent building control and Fault Detection and Diagnosis (FDD), because interpreting metadata still requires rigorous, manual mapping by experts who are intimately familiar with each building's configuration [3, 21, 22].

To address these interoperability challenges, standardized building ontologies have emerged, with the Brick schema [2] being a prominent example. Brick defines a common vocabulary and a set of standardized relationships that model dependencies among building subsystems [2]. This uniform representation transforms diverse BAS data into a structured graph format that supports the portability of advanced analytics and applications [11].

Despite these benefits, adoption is hampered by a steep learning curve required to create and query these graphs. The domain experts closest to building systems (e.g., facility managers and operators) rarely have the skills to query Brick-based knowledge graphs (KGs) using languages like SPARQL [14], leaving rich semantic data effectively inaccessible to the people who need it most [4]. Large Language Models (LLMs) could offer a promising natural language interface to complex building data, but accuracy, repeatability, and reliability are concerns. Mulayim et al. [21] indicate that LLMs often fail to strictly adhere to domain ontologies, hallucinating invalid classes or overlooking site-specific modeling choices in favor of generalized structures. These behaviors limit their utility for ontology-driven tasks.

Knowledge Graph Question Answering (KGQA) aims to bridge this gap by enabling users to pose natural language questions that are translated into executable queries over Knowledge Graphs (KGs) [17]. While general-knowledge KGQA has advanced [8, 19, 27], research on building ontologies remains limited, leaving its potential underexplored. Recently, BuildingQA [20], the first standardized benchmark dataset for KGQA in building ontologies, was released, representing a valuable step toward broadening the field. The paper encourages the research community to utilize the dataset to evaluate and refine new KGQA methods tailored to building semantics. To leverage building operational data, we need scalable KGQA methods grounded in building ontologies that generalize across diverse topologies without site-specific fine-tuning.

In this paper, we present BrickQA, an LLM-based, schema-guided framework optimized for building ontologies (Figure 1). By combining the interpretive power of LLMs with strict ontology constraints and inline validation, BrickQA generates reliable SPARQL without extensive task-specific training and supports end-to-end exploration and analytics over Brick-modeled datasets. It enables domain experts to interact with large databases using natural language while reducing hallucinations and schema violations, effectively bridging the gap between technical complexity and operational insight by substantially lowering the barrier to accessing building semantic models.

The main contributions of this paper are:

- **Dynamic Knowledge Graph Exploration for Building Informatics:** We present BrickQA, a schema-guided agent tailored for building ontologies that streamlines query decomposition, iterative dynamic exploration, and inline validation to navigate multi-hop Brick graphs. BrickQA specifically optimizes dynamic exploration to verify complex structural relationships in building topologies. By prioritizing local graph neighborhoods over exhaustive static context, it demonstrates robust adherence to heterogeneous facility structures while minimizing the need for site-specific fine-tuning.
- **Empirical Robustness and Cost-efficiency on Standardized Benchmarks:** BrickQA maintains structural fidelity across heterogeneous buildings and remains resilient to ambiguous queries, without site-specific fine-tuning. On BuildingQA, it delivers a 0.291–0.355 absolute F1 improvement over competitive agentic baselines while achieving 3×–12.7× higher cost-efficiency across multiple buildings and schema-visibility settings. Under more complex and ambiguous conditions, BrickQA preserved structurally coherent answers while competing baselines showed marked degradation.
- **Agent for Operational Analytics in Building Knowledge Graphs:** BrickQA bridges the semantic-to-operational gap by extending KGQA from simple entity retrieval to end-to-end facility diagnostics. A specialized Query Decomposition layer captures temporal constraints and aggregation operations, enabling multi-hop operational queries over time-series (e.g., sliding windows, counts, extrema). In a practical case study, the agent executed such analytics with a 90% success rate, empowering domain experts to derive insights without requiring deep technical knowledge of semantic web technologies or programming.

The rest of this paper reviews background in Section 2, details the BrickQA architecture in Section 3, evaluates its performance in Section 4, discusses the findings in Section 5, and concludes the paper in Section 6.

2 Background

2.1 Data Ontologies

A data ontology provides a formal, machine-processable conceptualization of a given domain, utilizing semantic modeling to ensure consistency and enable sophisticated reasoning over complex datasets [16]. Unlike simple databases or ad-hoc naming conventions, ontologies facilitate critical use cases by abstracting physical assets into a shared digital understanding. These include semantic interoperability, which ensures distinct subsystems (e.g., HVAC, lighting) exchange unambiguous information, and automated reasoning, which enables systems to derive new knowledge without explicit hard-coding. Furthermore, they support lifecycle management by maintaining data consistency across design, construction, and operational phases, effectively mitigating data loss [16].

To operationalize these concepts in the built environment, standardized schemas such as Brick [2], Project Haystack [23], and the emerging ASHRAE 223 standard [1] have been developed. These schemas utilize the Resource Description Framework (RDF) to model buildings as directed graphs. Specifically, Brick defines a standardized class hierarchy for physical, logical, and virtual assets (e.g., `Equipment`, `Point`) alongside a set of relationships (e.g., `feeds`,

`isPartOf`) [2]. This abstraction decouples data retrieval from physical system complexity, ensuring that queries remain valid across different equipment configurations. For instance, it allows retrieving all temperature setpoints on a specific floor without explicitly enumerating every individual room.

Interacting with these graph-based abstractions requires specific query languages. SPARQL [14] serves as the standard interface for RDF-based ontologies like Brick. Unlike SQL, which is designed for relational tables, SPARQL utilizes "triple patterns" (subject-predicate-object) to traverse the KGs [2]. This architecture enables powerful transitive queries, including tracing the upstream power supply of a faulty asset, which are computationally expensive or difficult to express in traditional table-based schemas. However, writing SPARQL queries demands a high degree of technical expertise and can be a labor-intensive process [21].

2.2 Knowledge Graph Question Answering (KGQA)

To apply Knowledge Graph Question Answering (KGQA) to building data effectively, it is necessary to distinguish between an ontology and a knowledge graph. An ontology specifies the domain schema, including entity types, properties, and relationships. By contrast, a KG instantiates this schema by encoding concrete facts, such as instances and their links, using the ontology's classes and relations [15]. In the building context, a KG captures spatial and functional relationships among equipment and data points (e.g., sensors, setpoints) of a specific building, enabling interoperability and automation for building management applications [2, 21].

KGQA facilitates access to large-scale knowledge graphs by mapping natural language questions to executable logical forms [17]. Current systems generally fall into three categories.

Subgraph Retrieval: These methods use vector embeddings to identify relevant graph sections [25, 26]. However, embedding massive KGs like Wikidata [29] is prohibitively expensive. Approaches often limit the scope to tiny subgraphs (e.g., 0.01% of Wikidata), rendering them inapplicable where the full graph must be accessible [30].

Semantic Parsing: Traditional systems translate questions directly into logical forms (e.g., SPARQL) as a sequence-to-sequence task [33, 34]. Xu et al. [31] achieved state-of-the-art results by fine-tuning LLaMA with a modified SPARQL syntax. While high-performing, these parsers typically require extensive fine-tuning and struggle with dynamic, incomplete schemas.

LLM-based Graph Exploration: Agentic frameworks like ReAct [32] demonstrated leading performance by integrating reasoning and action. Building on this, Sun et al. [27] advanced the field by instructing LLMs to dynamically explore the graph. Subsequently, approaches such as SPINACH [19] and Dynamic-KGQA [8] have further evolved this paradigm, actively navigating the schema to construct queries rather than relying on static generation.

Despite these advancements in active schema navigation, applying such LLM-based frameworks to KGQA in the building domain remains underexplored. While numerous KGQA benchmarks with logical forms have been introduced over the past decade to advance the state of the art in open domains [28, 29, 35], building KGs differ markedly from general-purpose KGs (e.g., Wikidata [29]) due to their dense, repetitive structure and higher lexical complexity compared to open-domain tasks [20]. Labels are often opaque alphanumeric tags such as `AHU_1_SAT`, making it difficult to fully leverage LLMs' generalization capabilities. Critically, these graphs necessitate complex multi-hop reasoning, as relationships between entities (e.g., linking a sensor to a room) are rarely direct and often require traversing multiple intermediate equipment nodes.

Moreover, unlike open-domain KGs that benefit from large training corpora, building-specific data is scarce, necessitating zero-shot or few-shot generalization [21]. Furthermore, while building ontologies provide a standardized structure for building data, a knowledge graph for a specific building can still be constructed in various ways due to modeling freedom, despite attempts to enforce guidelines and constraints [21]. Thus, model-awareness is a key characteristic required for KGQA grounded in building ontologies. As a result, LLM-based KGQA frameworks designed for open-domain KGs frequently fail to generate valid SPARQL in this setting. Mulayim et al. [20] introduced the first standardized benchmark for building-domain KGQA, showing that even top-performing methods struggle with the domain's lexical complexity, with the best baselines achieving a mean row matching F1 of 0.38.

To address this gap, BrickQA extends the agentic paradigm by introducing domain-specific iterative reasoning that explicitly optimizes multi-hop traversal within the Brick schema. By embedding model-awareness and ensuring structural fidelity, BrickQA successfully navigates these challenges where general-purpose agents often encounter performance degradation.

3 BrickQA

We introduce BrickQA, a schema-guided framework bridging natural language and building data via SPARQL. As shown in Figure 1, the architecture orchestrates structured decomposition, iterative generation, and inline validation tailored to the unique complexities of building informatics. Departing from static prompting, it employs dynamic schema exploration to navigate the ontology, ensuring executable and semantically aligned queries.

Query Decomposition: The goal of the decomposition layer is to ground natural-language questions in a structured representation that guides query generation and reduces schema errors and time-series-related hallucinations. It pre-processes the natural-language query to extract four components:

- **Query Intent** (e.g., retrieving sensor values, listing equipment, or comparing measurements)
- **Sensor Linking**, which maps natural-language expressions like "room temp" to ontology classes like `brick:Zone_Air_Temperature_Sensor`
- **Temporal Constraints**, which classify requirements into specific SPARQL patterns (e.g., sliding windows, absolute dates) to mitigate time-series-related hallucinations [10]
- **Operations**, which identify aggregation functions (e.g., `AVG`, `MAX`) for analytical queries and handles conditional logic, such as comparing values between multiple entities.

This structured representation provides the grounding context for subsequent reasoning. A concrete example of this decomposition is illustrated in the middle box of Figure 1.

Beyond providing this initial grounding context, the typed output also anchors BrickQA's recovery strategy in subsequent iterations. The decomposed sensor set, operations, and temporal pattern serve as a persistent reference that the iterative generation and inline query validation steps consult whenever the generated SPARQL must be diagnosed or revised across iterations.

To avoid unnecessary overhead, simple queries bypass the decomposition layer. Unless a query exceeds 12 words or includes multi-sensor indicators, comparisons, or conditionals, it proceeds directly to iterative SPARQL generation.

Iterative SPARQL Generation: The core reasoning engine is a ReAct-based [32] agent that constructs queries through a *Thought-Action-Observation* loop. At each timestep t, the agent generates a reasoning trace (Thought τ_t), selects an executable function (Action a_t), and processes the returned results (Observation o_t). The agent state $\mathcal{S}_t$ maintains the full history of these interactions. To perform the Action a_t, the agent is equipped with five tools adapted from SPINACH [19] for the Brick Schema domain:

- **`search_brick(term)`**: Search for Brick entities matching a text query
- **`get_brick_entity(id)`**: Retrieve properties and relationships of an entity
- **`get_property_examples(prop)`**: Show usage examples of a property
- **`execute_sparql(query)`**: Execute a SPARQL query against the Brick graph
- **`stop()`**: Mark the current query as the final answer

The following example demonstrates a single iteration of this reasoning loop:

- **Question**: "What is the average airflow of VAV 2-01?"
- **Thought (τ_t)**: I need to find the sensor ID associated with VAV 2-01 to query its airflow data. I will search for the VAV entity first.
- **Action (a_t)**: `search_brick("VAV 2-01")`
- **Observation (o_t)**: `[{"id": "bldg:VAV_2-01", "type": "brick:Variable_Air_Volume_Box"}]`

While generating SPARQL, this iterative loop also serves as BrickQA's recovery channel for issues that cannot be repaired deterministically by the inline validation step and instead require the LLM to generate a revised response. Empty result sets are recorded in the observation log, allowing the agent to revise its structural path in the next iteration. Additionally, malformed LLM outputs trigger a single error-aware retry before the process resumes. The loop continues in this way until the agent successfully executes the `stop()` action or reaches a predefined maximum number of iterations, which is set to 15 in this study.

BrickQA employs dynamic schema exploration via `search_brick` to overcome the scalability limits of static injection. Injecting large building knowledge graphs is impractical due to prohibitive token costs and context window constraints. Furthermore, large contexts suffer from the "Lost in the Middle" effect [18], where retrieval accuracy degrades, despite existing mitigation strategies [24, 36]. Our approach mitigates these overheads, scaling to large buildings without increasing prompt size.

To guide the query generation process, BrickQA incorporates a concise building description, generated by an LLM from a subset of the TTL file, into the system prompt. This strategy drastically reduces token consumption compared to exhaustive triple injection, while explicitly capturing site-specific conventions, such as naming regexes (e.g., `.*_nor`) and traversal paths. By distilling the graph structure into actionable guidelines, the agent can generalize to unseen entities without needing to observe a large number of sample triples.

Inline Query Validation: To ensure execution safety, BrickQA implements a validation layer that intercepts the `execute_sparql` action, performing rule-based checks before execution:

- **Syntax**: corrects common namespace and format errors.
- **Temporal**: checks alignment with extracted time constraints.
- **Sensor Coverage**: verifies all required data streams are referenced.
- **Join**: checks timestamp alignment in multi-sensor queries.

Detected issues are surfaced as warnings in the observation log rather than being corrected automatically. These warnings are fed back into the iterative generation loop, enabling the agent to perform informed self-correction in the next iteration by grounding its subsequent action in the flagged constraints.

4 Evaluation

Our experimental evaluation is designed to assess both domain generalizability and practical operational utility. We first perform a benchmark evaluation using the BuildingQA dataset [20] to compare our approach against the ReAct agent baseline established in prior work. In addition to standard accuracy metrics, we conduct a comprehensive cost vs. performance analysis to highlight the economic efficiency of our dynamic exploration strategy. To ensure a fair comparison during the benchmark evaluation, we disabled the *Temporal Constraints* and *Operations* modules of the decomposition layer as these capabilities fall outside the scope of the standard benchmark.

Subsequently, we present a case study utilizing a custom set of practical questions designed to rigorously evaluate the framework's temporal reasoning and analytical precision. This case study demonstrates the comprehensive capabilities of our framework, highlighting its proficiency in complex aggregation and time-series operations, such as counting and extrema analysis, that are indispensable for routine facility management. Consequently, this evaluation bridges the gap between theoretical parsing and actionable operational insights.

4.1 Experimental Setup

4.1.1 Datasets. Table 1 summarizes the structural complexity (triples, entities, and key Brick instances) of the distinct datasets employed for the benchmark evaluation and operational case study.

Standardized Benchmark (BuildingQA): For the standardized benchmarking, we utilize the BuildingQA dataset, a resource constructed to address the scarcity of domain-specific evaluation tools [20]. Its creation involved collecting real-world SPARQL queries and building models from domain experts to ensure operational relevance. To test linguistic robustness, these expert-authored queries were augmented via LLMs to produce five paraphrases per query, ranging from the most direct template translations (`LLM_1`) to the

Table 1: Brick schema complexity across benchmark buildings. (Ents: Entities, Pts: Points, Equip: Equipment, Locs: Locations).

Dataset	Triples	Ents.	Pts.	Equip.	Locs.
DFLEX [6]	629	221	71	4	6
TUC [9]	1,855	558	198	1	104
BLDG11 [11]	62,578	13,606	1,292	488	796
LBNL FDD FCU [12]	67	35	29	5	1

most ambiguous, high-level inquiries (`LLM_5`). Given that our framework is designed specifically for the Brick schema, we excluded one building (BLDG59) as it adheres to the ASHRAE 223 standard, and utilized the remaining three buildings (DFLEX, TUC, and BLDG11) from this benchmark for our evaluation (Table 1).

Operational Analytics Dataset (LBNL FDD FCU): For the operational analytics case study, we utilize the Lawrence Berkeley National Laboratory (LBNL) FDD Fan Coil Unit (FCU) Dataset [12]. The dataset includes 29 data points logged at 1-minute intervals across 49 fault-free and faulty scenarios (e.g., sensor bias, valve leakage, stuck dampers). To ensure computational simplicity during SPARQL execution, we resampled the raw 1-minute data to an hourly resolution. This compact yet rich time-series operational data is suitable for evaluating our framework's temporal and analytical reasoning capabilities.

4.1.2 Metrics. We adopt the hierarchical framework detailed in BuildingQA [20] and report performance across three progressively stricter metrics:

- **Entity Set F1**: Evaluates entity linking by measuring the retrieval of unique values per column, independent of order or relational pairing.
- **Row Matching F1**: Assesses topological correctness by requiring entire rows (entity pairs) to match the ground truth after optimal column alignment.
- **Exact Match F1**: The strictest measure, equivalent to execution accuracy, requiring a perfect result set match without any alignment.

4.1.3 Baselines. All tasks involving LLMs were performed using Gemini 2.5 Flash [7]. To evaluate the impact of schema context, we adopt the three configurations defined in BuildingQA [20]:

- **No Knowledge Graph (NoKG):** A setting where the model receives no initial schema triples. In this setting, BrickQA relies entirely on dynamic exploration via `search_brick()` to retrieve context on demand.
- **100 Triples (100T):** A sparse setting where the first 100 triples from the building's TTL file are injected directly into the prompt.
- **5000 Triples (5000T):** A dense setting where the first 5000 triples from the building's TTL file are injected to provide a comprehensive structural overview.

Although BrickQA is designed for the No Knowledge Graph (No KG) setting to ensure scalability to arbitrary building sizes without context saturation (as discussed in Section 3), we evaluate it across all three configurations to ensure a direct paired comparison with the results reported in [20]. This approach further allows us to quantify the marginal benefit or potential degradation of combining static context with dynamic exploration, particularly given that excessive context can sometimes impair LLM reasoning.

Table 2: Comparison of BrickQA and ReAct across three buildings under three graph context sizes: No Knowledge Graph, 100 Triples, and 5000 Triples. The highest score in each pair is bolded. "Total" represents the macro-average.

	Entity Set F1		Row Matching F1		Exact Match F1	
Dataset	BrickQA	ReAct	BrickQA	ReAct	BrickQA	ReAct
No Knowledge Graph						
DFLEX	**0.504**	0.028	**0.347**	0.014	**0.042**	0.000
TUC	**0.750**	0.000	**0.633**	0.000	**0.567**	0.000
BLDG11	**0.455**	0.306	**0.398**	0.176	**0.110**	0.000
Total	**0.570**	0.111	**0.460**	0.063	**0.239**	0.000
100 Triples						
DFLEX	**0.542**	0.296	**0.472**	0.264	**0.167**	0.000
TUC	**0.950**	0.262	**0.933**	0.083	**0.867**	0.000
BLDG11	**0.409**	0.370	**0.361**	0.231	**0.136**	0.000
Total	**0.634**	0.309	**0.589**	0.193	**0.390**	0.000
5000 Triples						
DFLEX	**0.787**	0.675	**0.676**	0.593	**0.292**	0.000
TUC	**1.000**	0.865	**0.989**	0.710	**0.889**	0.000
BLDG11	**0.443**	0.419	**0.405**	0.238	**0.130**	0.000
Total	**0.743**	0.653	**0.690**	0.514	**0.437**	0.000

4.2 Evaluation on BuildingQA

4.2.1 Quantitative Analysis. The quantitative evaluation on the BuildingQA benchmark confirms that BrickQA's dynamic exploration strategy significantly outperforms agentic workflows with static prompting across diverse building topologies and schema availability settings. Table 2 summarizes the comparative performance against the ReAct baseline, while Figures 2 and 3 provide granular breakdowns by dataset complexity and query type, respectively. Across all evaluation scenarios, BrickQA achieves a substantial average absolute F1 improvement of 0.291, 0.323, and 0.355 in Entity Set, Row Matching, and Exact Match metrics, respectively. Overall, the results validate our core hypothesis: decoupling semantic reasoning from static context injection via dynamic exploration allows for superior scalability, particularly in large-scale environments like BLDG11, where complete knowledge graph indexing is computationally prohibitive. In the following analysis, we dissect these findings through three complementary metrics to assess retrieval recall (i.e., Entity Set F1), logical traversal accuracy (i.e., Row Matching F1), and syntactic precision (i.e., Exact Match F1).

The comparison of Entity Set F1 (hereafter Entity F1) scores in Table 2 reveals distinct behaviors regarding graph dependency and scalability. BrickQA demonstrates remarkable robustness in No KG, achieving a total Entity F1 of 0.570 compared to ReAct's 0.111. This indicates that BrickQA successfully internalizes the Brick schema semantics, allowing it to identify correct entity types even without

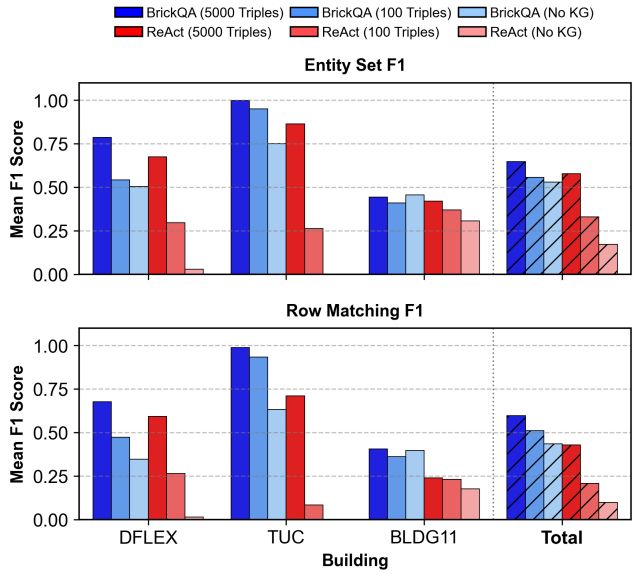

Figure 2: Performance comparison between BrickQA and ReAct across three buildings under varying schema contexts.

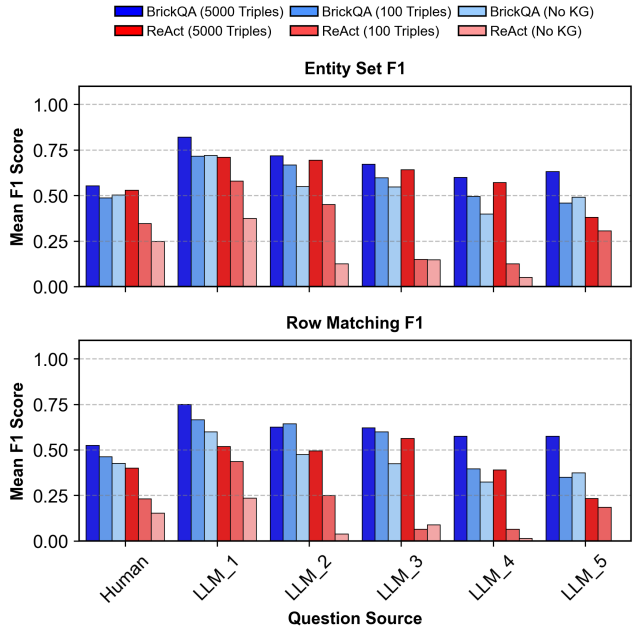

Figure 3: Performance comparison between BrickQA and ReAct across human-authored queries (Human) and five levels of LLM-generated paraphrases (LLM_1 to LLM_5).

explicit graph traversal. Crucially, we observe that performance is heavily influenced by the graph complexity outlined in Table 1. On smaller datasets, BrickQA achieves high entity retrieval accuracy in the 5,000-triple setting (5000T), with an Entity F1 of 1.000 for TUC and 0.787 for DFLEX. In contrast, on the larger BLDG11 dataset (62,578 triples), performance naturally decreases for both systems due to the expanded search space. However, BrickQA maintains a robust advantage, recording Entity F1 scores between 0.409 and 0.455, whereas ReAct stays between 0.306 and 0.419.

Beyond simple entity retrieval, the Row Matching F1 (hereafter Row F1) metric offers a more critical assessment of the model's ability to correctly traverse graph relationships. BrickQA consistently outperforms ReAct on this metric across all buildings and all settings. This performance gap is particularly significant on the larger BLDG11 dataset, where BrickQA achieves scores ranging from 0.361 to 0.405, compared to ReAct's lower performance of 0.176 to 0.238.

Finally, the Exact Match F1 metric provides supplementary insight into syntactic precision. ReAct scores 0.000 across all datasets because it fails to replicate the strict column ordering and specific SPARQL syntax of the ground truth, as reported in [20]. Conversely, BrickQA achieves a total Exact Match F1 of 0.239 in the No KG setting, which rises to 0.437 in 5000T. While Row F1 could be the more practical measure of analytical success, these non-zero Exact Match scores highlight BrickQA's superior control over query syntax and output formatting compared to the baseline agent.

Crucially, BrickQA demonstrates superior scalability on the large-scale BLDG11 dataset, where its performance in the No KG setting (Entity F1: 0.455, Row F1: 0.398) consistently exceeds even the best-case ReAct baseline equipped with 5,000 triples (0.419 and 0.238, respectively). By decoupling logical reasoning from static instance availability, BrickQA effectively navigates vast search spaces without the overhead of dense context injection, presenting a more data-efficient and practical solution for real-world deployments where complete indexing is infeasible.

Additionally, we observe a divergence in the impact of context scaling in BrickQA. While performance on smaller datasets (TUC and DFLEX) improves monotonically with increased triple counts, BLDG11 exhibits an interesting anomaly where Entity F1 and Row F1 scores are lowest in the 100-triple (100T) setting (0.409 and 0.361, respectively). We attribute this degradation to the introduction of irrelevant context; for a massive graph exceeding 60,000 triples, injecting an arbitrary subset of the first 100 triples likely acts as distractor noise rather than a helpful signal. This sparse context appears to confound the agent, leading to lower performance compared to the No KG setting where the agent relies solely on dynamic exploration.

As illustrated in Figure 3, we further disaggregate performance by query source to evaluate resilience against linguistic variation. While a downward trend is observed for both approaches as query ambiguity increases, BrickQA demonstrates significantly greater stability. Specifically, in 5000T, the ReAct agent suffers a distinct collapse in Row F1, falling from 0.563 on `LLM_3` to 0.233 on `LLM_5`. In contrast, BrickQA exhibits only a marginal decline from 0.621 to 0.574. Remarkably, on `LLM_5` queries, the most ambiguous category, BrickQA without instance triples (i.e., no KG) outperforms the ReAct agent augmented with 5000 triples, achieving superior scores in both Entity F1 (0.490 vs. 0.380) and Row F1 (0.374 vs. 0.233). This confirms that for ambiguous inquiries, active schema exploration yields higher utility than passive context ingestion.

4.2.2 Cost vs. Performance Analysis. To evaluate practical deployment viability, we analyze the trade-off between query accuracy and API cost across all approaches in the BuildingQA benchmark. Figure 4 plots the mean Row F1 against the average per-question cost for each approach–model combination, with baseline values adopted from BuildingQA [20].

As shown in Figure 4, three BrickQA configurations occupy the upper-left Pareto frontier, demonstrating superior accuracy at a significantly lower cost. BrickQA (100T) attains a Row F1 of 0.510 at a mere $0.005 per question, while BrickQA (5000T) achieves the highest overall F1 of 0.597 at $0.026. Notably, even BrickQA (No KG), operating without any instance triples, reaches an F1 of 0.435 at $0.006, outperforming all ReAct configurations except for the ReAct (5000T) variant paired with o3-mini. Although that o3-mini configuration is the most competitive baseline with an F1 of 0.481, it costs $0.064 per question; this is 12.7× the cost of BrickQA (100T) for a lower F1 score. Furthermore, the most cost-effective baseline, ReAct (5000T) with Gemini 2.5 Flash, achieves 0.428 at $0.015, which still requires 3× the cost of BrickQA (100T) while yielding a lower F1 score.

A closer examination of token utilization reveals the mechanism behind BrickQA's cost efficiency. Despite consuming more tokens per query than ReAct, specifically 2.9× more in the No KG setting (35,516 vs. 12,120 avg. tokens/query) and 2.0× more in 5000T (170,689 vs. 84,726 avg. tokens/query), BrickQA achieves lower costs due to a fundamentally different token distribution. Leveraging a history accumulation strategy, BrickQA aggregates the full trajectory of prior thoughts, actions, and observations into the prompt context at each iteration. This design effectively offloads the maintenance of reasoning state to the input, allowing the agent to perform complex self-correction and strategy refinement by reading its history rather than generating verbose chain-of-thought rationales. Consequently, BrickQA maintains minimal completion outputs, yielding prompt-to-completion ratios of 21:1 to 168:1, whereas ReAct produces 5–10× more completion tokens (e.g., 12,525 vs. 1,894 in 100T) to sustain its reasoning context.

Since Gemini 2.5 Flash, like most other LLM APIs, charges 4× more for completion tokens ($0.60/M) than prompt tokens ($0.15/M), this shift from generative reasoning to context-based retrieval translates directly to cost savings. For example, in 100T, BrickQA's cost breakdown is 88% prompt ($0.0044) and 12% completion ($0.0006), totaling $0.005 per question. ReAct inverts this ratio: 15% prompt ($0.0011) and 85% completion ($0.0063), totaling $0.0074, which is 48% more expensive despite using 41% fewer total tokens. This demonstrates that BrickQA's prompt-heavy design, which leverages accumulated context as a cost-effective working memory, is economically superior under modern pricing structures that penalize completion-heavy generation.

4.2.3 Error Analysis. To diagnose the root causes of performance degradation, Table 3 categorizes failures (defined as F1 < 0.3) by structural (*Arity Mismatch*) and semantic (*Relationship Error*) patterns. Two dominant trends emerge. First, *Relationship Errors* serve as the primary bottleneck for logical traversal in complex environments; in BLDG11, they account for 100% of Row failures across all settings. This indicates that even when entities are retrieved, BrickQA struggles to correctly identify the specific semantic paths

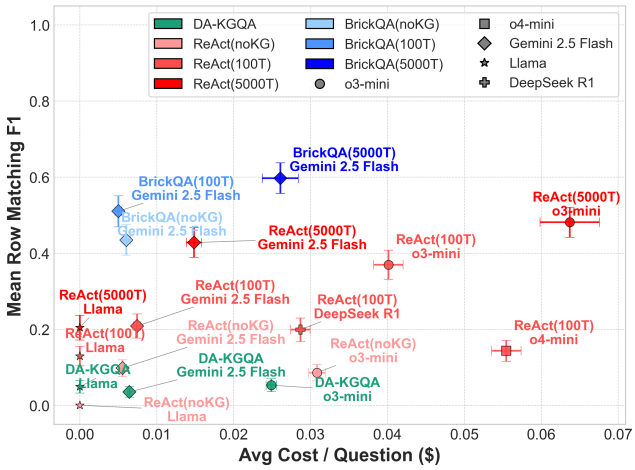

Figure 4: Cost vs. performance trade-off across all approaches on BuildingQA. Each point represents an approach–model combination; error bars indicate standard error.

connecting them in a dense graph. Second, The results suggest that partial context can be detrimental, particularly in large graphs, when it lacks semantic relevance to the query. In the BLDG11 dataset, the proposed system performed worst in 100T, reaching a failure rate of 55%, higher than having no knowledge graph at all (49%). As detailed in Section 4.2.1, the performance dip for BLDG11 in 100T might stem from a scale mismatch. Providing only a sparse, arbitrary subset of triples from a large graph introduces distractor noise that misleads BrickQA. This results in a high rate of *Arity Mismatches* (95% of failures), leading to poorer outcomes than having no external context at all. To illustrate these failure mechanisms, we present representative error cases derived from actual benchmark queries.

Error Pattern 1: Structural Ambiguity (Arity Mismatch). The dominant failure mode is *arity mismatch*, where the BrickQA agent retrieves a different number of projection variables (columns) than the ground truth. This pattern accounts for 75–100% of Entity F1 failures and 36–100% of Row F1 failures, effectively creating a bottleneck where structural disagreement precludes semantic evaluation. As shown below, this often stems from the implicit nature of natural language:

BLDG11 – MORTAR_001, Q1 (Human):
"What are the brick points in this graph?"

 # Generated (1 column):

```
?point rdf:type/rdfs:subClassOf* brick:Point .
```

 # Ground Truth (2 columns):

```
?point rdf:type/rdfs:subClassOf* brick:Point .
?point rdf:type ?point_type .
```

Although the generated query retrieves 1,257 point URIs, the arity mismatch (1 column vs. 2 columns) causes both Entity F1 and Row F1 to drop to zero. This highlights a fundamental challenge in text-to-SQL/SPARQL tasks: user intent is often underspecified regarding

output format, leading to valid interpretations that statistically fail against rigid ground truth benchmarks.

Error Pattern 2: Semantic Traversal Failures. Beyond structural format, the second major challenge lies in navigating multi-hop relationships within complex topologies. This error type is disproportionately prevalent in the dense BLDG11 dataset (accounting for 76–100% of failures). While models often correctly identify the target entity types (e.g., VAVs and Zones), they struggle to reconstruct the specific relationship chains required to link them.

For instance, in the representative failure case below (observed in 100T), BrickQA must deduce that sensors are not properties of the zone itself, but are hosted by the VAV unit feeding that zone. This requires synthesizing a multi-hop path that bridges the equipment (VAV) with its location (Zone) and its components (Sensors):

DFLEX – DFLEXLIBS_001, Q1 (Human):
"For each HVAC zone in the building, what are the timeseries IDs for the available zone temperature sensor' points, min and max temperature setpoints, temperature setpoints (single, heating/cooling or occupied/unoccupied), occupancy sensors and commands, and any related VAV damper, discharge temperature, and reheat command points?"

 # Generated (Incorrect Shortcut):

```
?zone brick:hasPoint ?sensor . # No direct link exists
```

 # Ground Truth (Required Multi-hop Path):

```
?vav brick:feeds ?zone . # 1. First, locate equipment
?vav brick:hasPoint ?sensor . # 2. Then, find sensors
```

Failures here typically involve hallucinating direct relationships (e.g., `?zone hasPoint ?sensor`) where intermediate nodes are required. This suggests that while BrickQA's schema exploration effectively identifies node concepts, reasoning over long-range dependency paths remains a complexity barrier in large-scale graphs.

Error Pattern 3: Semantic Granularity Mismatch. A subtle yet significant subset of errors involves cases with moderate-to-high Entity F1 but near-zero Row F1. This phenomenon occurs when the agent interprets the query at a different semantic granularity than intended, retrieving related but incorrectly scoped entities. This misalignment is demonstrated in the following example:

BLDG11 – MORTAR_002, Q7 (LLM_3):
"Which HVAC components provide setpoints for regulating zone air temperature and what are those setpoints called?"

 # Generated (returns setpoint types + AHUs):

```
SELECT ?equipment ?setpoint_type WHERE {
  ?setpoint_instance rdf:type/rdfs:subClassOf* ?
       setpoint_type .
  VALUES ?setpoint_type { brick:Air_Temperature_Setpoint ...
        }
  ?equipment brick:hasPoint ?setpoint_instance . }
```

 # Ground Truth (returns specific setpoint instances + VAVs):

```
SELECT ?sp ?equip WHERE {
  ?sp rdf:type/rdfs:subClassOf* brick:
       Zone_Air_Temperature_Setpoint .
  ?equip brick:hasPoint ?sp . }
```

Table 3: Distribution of BrickQA error patterns grouped by failure criteria.

Condition / Pattern	DFLEX	TUC	BLDG11
No Knowledge Graph			
— *Entity Set F1* < 0.3 —			
Arity Mismatch	18 (100%)	3 (75%)	35 (95%)
Relationship Error	6 (33%)	2 (50%)	34 (92%)
Total Failures	*18 (50%)*	*4 (13%)*	*37 (49%)*
— *Row Matching F1* < 0.3 —			
Arity Mismatch	18 (100%)	4 (36%)	35 (78%)
Relationship Error	18 (100%)	11 (100%)	45 (100%)
Total Failures	*18 (50%)*	*11 (37%)*	*45 (59%)*
100 Triples			
— *Entity Set F1* < 0.3 —			
Arity Mismatch	15 (94%)	1 (100%)	40 (95%)
Relationship Error	7 (44%)	0 (0%)	40 (95%)
Total Failures	*16 (44%)*	*1 (3%)*	*42 (55%)*
— *Row Matching F1* < 0.3 —			
Arity Mismatch	16 (94%)	2 (100%)	40 (85%)
Relationship Error	17 (100%)	2 (100%)	47 (100%)
Total Failures	*17 (47%)*	*2 (7%)*	*47 (62%)*
5000 Triples			
— *Entity Set F1* < 0.3 —			
Arity Mismatch	7 (100%)	0 (N/A)	36 (95%)
Relationship Error	5 (71%)	0 (N/A)	29 (76%)
Total Failures	*7 (19%)*	*0 (0%)*	*38 (50%)*
— *Row Matching F1* < 0.3 —			
Arity Mismatch	8 (89%)	0 (N/A)	38 (84%)
Relationship Error	8 (89%)	0 (N/A)	45 (100%)
Total Failures	*9 (25%)*	*0 (0%)*	*45 (59%)*

Note: Pattern percentages are of failures within each criteria (can overlap). Total Failures percentage is of all questions. N/A = no failures.

The generated query operates at the wrong level of abstraction: it returns AHU-level equipment (e.g., `AHU01`) paired with setpoint *class types* (e.g., `Air_Temperature_Setpoint`), whereas the ground truth expects VAV-level equipment (e.g., `VAVRM1018`) paired with specific setpoint *instances* (e.g., `Zone_Air_Temp_Setpoint`). While 40% of entities overlapped due to shared URIs in the knowledge graph (yielding Entity F1 = 0.489), none of the 232 generated rows matched any of the 222 ground truth rows because the semantic scope was fundamentally misaligned. This pattern validates the necessity of the Row F1 metric: partial entity overlap alone is insufficient for guaranteeing correct structured knowledge retrieval.

4.3 Case Study: Operational Analytics

To validate BrickQA's utility in real-world facility management, we deployed the system on the LBNL FDD dataset [12] using 10 diverse queries designed to mimic the workflows of building operators. As shown in Table 4, the system achieved a 90% success rate (9/10), demonstrating robust capability across operational domains. In the area of fault detection, it effectively identified issues such as simultaneous heating and cooling or verified valve positions during specific fan states (e.g., Questions 3, 8). For comfort analysis, the

Table 4: List of questions evaluated on the LBNL FDD dataset and the success status of BrickQA.

#	Natural Language Query	Success
1	Find periods in the last 24 hours where the room temperature was higher than the discharge air temperature.	✓
2	Show me the outdoor air humidity on September 20th when the cooling coil water flow was zero.	✓
3	What was the heating valve position yesterday when the fan status was off?	×
4	Retrieve the return air temperature on September 18th where the mixed air damper position was greater than 80%.	✓
5	Compare the room temperature and the cooling setpoint between 9 AM and 5 PM on September 1st.	✓
6	Calculate the average difference between discharge air temperature and room temperature on August 25th.	✓
7	Find the maximum outdoor air temperature observed during the last 24 hours.	✓
8	Identify time intervals on August 25th where the heating valve and cooling valve were both open simultaneously.	✓
9	Count how many times the room temperature exceeded the cooling setpoint on September 10th.	✓
10	How long was the room warmer than the cooling setpoint over the past two days?	✓

system successfully evaluated zone temperature deviations against setpoints over sliding temporal windows (e.g., Questions 1, 5, 10). Furthermore, it showed proficiency in energy profiling by correlating internal system loads with external environmental factors, including outdoor air temperature and humidity (e.g., Questions 2, 7).

Specifically, the successful execution of Question 10 (visualized in Figure 1) confirms the *Query Decomposition* layer's ability to handle compositional complexity. Unlike simple retrievals, this query requires a multi-stage reasoning process: resolving the specific zone, retrieving historical time-series data for both temperature and setpoint, and applying a temporal difference function ("How long..."). The ability to automate such high-level analytical intents bridges the gap between semantic parsing research and actionable building diagnostics.

Conversely, the single failure in Question 3 highlights a critical challenge: the mismatch between the user's abstract intent and the specific data structure. While the user conceptually queried for a discrete binary state ("fan status off"), the underlying graph topology characterized fan operation solely through a continuous variable, `brick:Speed_Status`. This case illustrates a gap where the semantic abstraction in the graph (i.e., class names) does not perfectly align with its operational semantics (i.e., how the equipment actually functions). While BrickQA's iterative generation component improves success rates by enabling the system to fall back and explore alternative paths when a sensor is missing, it remains insufficient to bridge this gap, as it rigidly searched for a non-existent `brick:On_Off_Status` instead of inferring that a zero speed value functionally implies an "Off" state, resulting in a false negative. This failure mode underscores the necessity for proxy reasoning in future iterations, where models must look beyond direct lexical mapping to interpret continuous sensor readings as proxies for discrete operational states.

4.4 Limitations

While BrickQA demonstrates the efficacy of schema-guided reasoning for the exploration of building KGs, our study operates within specific constraints that highlight critical avenues for future refinement. We identify three primary limitations.

Evaluation Scope and Model Bias: Our evaluation currently relies on a single experimental run with a proprietary LLM (i.e., Gemini 2.5 Flash [7]), which may not fully capture the stochastic variance inherent in probabilistic generation. Relying on a sole model also complicates decoupling architectural contributions from model-specific priors. Future work should prioritize multi-trial evaluations across diverse LLMs, including open-source alternatives, to rigorously quantify performance variance and ensure generalizability.

Iterative Latency and Computational Cost: While BrickQA outperforms the baselines in token efficiency, its iterative generation inherently creates a "chatty" interaction pattern, incurring significant latency and token costs, particularly for complex multi-hop queries. While this step-by-step validation mitigates hallucinations, the resulting precision-latency trade-off remains a bottleneck for real-time applications. Future optimizations, such as subgraph caching and parallelized execution, are necessary to reduce this overhead.

Gap Between Static Benchmarks and Operational Reality: The scarcity of high-fidelity building KGs restricted our evaluation to relatively small, sanitized datasets. While the BuildingQA benchmark [20] provides a standardized foundation, these models may not fully capture the complexity of large-scale operational deployments across diverse building types and climates. Furthermore, real-world building knowledge graphs are often incomplete or inconsistently modeled [21, 22] , presenting a significant hurdle for robust semantic parsing within this domain. This scarcity also necessitated a limited set of ad-hoc queries for our temporal analytics case study. While illustrative, evaluating a more extensive suite of operational queries and establishing standardized, community-driven benchmarks are essential next steps to advance building KGQA systems. Validating BrickQA on diverse, large-scale implementations will be a valuable direction to confirm its scalability and resilience in fragmented data environments.

5 Discussion

Our findings compel a re-evaluation of how LLMs should interact with complex cyber-physical systems such as buildings. In contrast to approaches relying on extensive context loading, our results demonstrate that active navigation of the graph topology serves as the cornerstone of effective semantic interpretation, particularly in the building domain where massive graphs and multi-hop relationships are prevalent.

Extending to Heterogeneous Semantic Standards: While our evaluation centers on the Brick schema, one of the key features

of BrickQA is its schema-agnostic architecture. It operates by dynamically traversing the graph and validating syntax against the provided ontology definition in real-time. This structural flexibility suggests that the framework is not confined to Brick, but is readily applicable to other standards, such as ASHRAE 223 [1]. This is particularly critical as the schemas of nascent standards may not be represented in the LLM's massive training data, necessitating a systematic approach to verify and overcome such data gaps. Given that the built environment is increasingly defined by a plurality of semantic models [22], the ability of an LLM agent to interpret diverse ontologies without retraining is a critical prerequisite for universal interoperability. Future work will focus on benchmarking BrickQA across these heterogeneous standards to evaluate its robustness in diverse modeling contexts.

Integrating Dynamic Exploration and Retrieval Paradigms: While our study demonstrates that dynamic schema exploration effectively substitutes for static schema injection, cost-performance analysis reveals that augmenting BrickQA with a minimal graph subset (100 triples) yields the optimal balance between accuracy and cost. This configuration is over 5× more cost-effective than the 5000-triple setting while achieving superior accuracy compared to the No KG baseline. However, the persistence of *Relationship Errors* highlights that implicit topological reasoning remains a hurdle; LLMs struggle to synthesize multi-hop dependencies without explicit path guidance. Consequently, future systems should evolve toward a hybrid approach that integrates schema-first reasoning with retrieval paradigms like Graph-based Retrieval-Augmented Generation (GraphRAG) [13]. By leveraging schema constraints for structural validity and GraphRAG for targeted instance retrieval, this synergy balances ontological rigor with contextual richness, maximizing precision without risking context saturation.

Transitioning to Interactive Disambiguation: The prevalence of *Arity Mismatch* errors validates the "Ambiguity of the Ground Truth" also discussed in BuildingQA [20]. This underscores a duality between scientific benchmarking and practical application. While static, reproducible test cases are essential for establishing performance baselines, operational reality necessitates interactive frameworks capable of navigating inherent underspecification. A statistical failure to align with a fixed ground truth often stems from a valid divergence in interpretation rather than an agent defect. To address this, the field should move toward interactive disambiguation agents that leverage iterative user feedback for refinement. By proactively engaging the user to resolve structural ambiguities, systems can incrementally reduce uncertainty.

Addressing Data Scarcity and Integrity: Scalable KGQA agents hold the potential to maximize the value of underutilized building data. However, this advancement is currently stifled by the disconnect between sanitized benchmarks and real-world systems, which must handle noisy data and ambiguous queries. While our case study demonstrated utility in simple fault detection, the field still lacks comprehensive benchmarks that capture the complexity of temporal diagnostics. To bridge this gap, community efforts are needed to simultaneously democratize data and formalize operational workflows. Yet, a fundamental bottleneck remains: open-source, updated semantic models are scarce, and their construction is labor-intensive. Furthermore, even existing models cannot be implicitly treated as ground truth due to frequent schema inconsistencies, the high degree of freedom in modeling equivalent systems, and the inherent heterogeneity of underlying physical infrastructure [20, 21]. Therefore, fostering open-source, community-driven initiatives to curate and validate building knowledge graphs is a prerequisite. Such an ecosystem will enable researchers to stress-test architectures against the messy reality of the built environment.

6 Conclusion

In this work, we introduced BrickQA, a schema-guided LLM framework that seamlessly translates natural language into executable SPARQL queries for building ontologies. By integrating query decomposition, dynamic traversal of local graph neighborhoods, and inline validation, BrickQA mitigates LLM hallucinations in SPARQL query generation and guarantees structural robustness without relying on model fine-tuning or exhaustive triple injection. Furthermore, a history-accumulation design improves cost-efficiency by shifting the computational burden from generation to retrieval, all while preserving strict SPARQL control. Empirically, BrickQA delivers substantial gains: on BuildingQA, it achieves absolute F1 improvements of 0.291–0.355 over ReAct baselines and is 3× to 12.7× more cost-effective. In a real-world operational case study, the agent executed temporal and aggregation analytics with a 90% success rate, underscoring a practical path from abstract semantic models to actionable facility insights.

Acknowledgments

This work was supported in part by the National Science Foundation (NSF) under Grant No. 2435567 and the Center for Integrated Facility Engineering (CIFE). This work was supported by the Assistant Secretary for Critical Minerals and Energy Innovation, Office of Building Technology of the U.S. Department of Energy under Contract No. DE-AC02-05CH11231. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the NSF or CIFE or DOE.

References

[1] ASHRAE. 2025. *ASHRAE Standard 223P: Semantic Data Model for Analytics and Automation Applications in Buildings.* Atlanta, GA, USA. https://open223.info/ Accessed: 2026-01-23.

[2] Bharathan Balaji, Arka Bhattacharya, Gabriel Fierro, Jingkun Gao, et al. 2016. Brick: Towards a Unified Metadata Schema for Buildings. In *Proceedings of the 3rd ACM International Conference on Systems for Energy-Efficient Built Environments (BuildSys '16)* (Palo Alto, CA, USA). Association for Computing Machinery, New York, NY, USA, 41–50. doi:10.1145/2993422.2993577

[3] Bharathan Balaji, Arka Bhattacharya, Gabriel Fierro, Jingkun Gao, et al. 2018. Brick: Metadata schema for portable smart building applications. *Applied Energy* 226 (Sept. 2018), 1273–1292. doi:10.1016/j.apenergy.2018.02.091

[4] Imane Lahmam Bennani, Anand Krishnan Prakash, Marina Zafiris, Lazlo Paul, et al. 2021. Query relaxation for portable brick-based applications. In *Proceedings of the 8th ACM International Conference on Systems for Energy-Efficient Buildings, Cities, and Transportation* (Coimbra, Portugal) *(BuildSys '21)*. Association for Computing Machinery, New York, NY, USA, 150–159. doi:10.1145/3486611.3486671

[5] Harry Bergmann, Cory Mosiman, Avijit Saha, Selam Haile, et al. 2020. *Semantic Interoperability to Enable Smart, Grid-Interactive Efficient Buildings.* Report OSTI ID: 1735554. Lawrence Berkeley National Laboratory. https://escholarship.org/uc/item/1325d5j3

[6] David Blum, Javier Arroyo, Sen Huang, Ján Drgoňa, et al. 2021. Building optimization testing framework (BOPTEST) for simulation-based benchmarking of control strategies in buildings. *Journal of Building Performance Simulation* 14, 5 (2021), 586–610. doi:10.1080/19401493.2021.1986574

[7] Gheorghe Comanici, Eric Bieber, Mike Schaekermann, et al. 2025. Gemini 2.5: Pushing the Frontier with Advanced Reasoning, Multimodality, Long Context, and Next Generation Agentic Capabilities. arXiv:2507.06261 [cs.CL] https://arxiv.org/abs/2507.06261

[8] Preetam Prabhu Srikar Dammu, Himanshu Naidu, and Chirag Shah. 2025. Dynamic-KGQA: A Scalable Framework for Generating Adaptive Question Answering Datasets. arXiv:2503.05049 [cs.CL] https://arxiv.org/abs/2503.05049

[9] Flavia de Andrade Pereira, Kyriakos Katsigarakis, Dimitrios Rovas, Marco Pritoni, et al. 2025. A semantics-driven framework to enable demand flexibility control applications in real buildings. *Advanced Engineering Informatics* 64 (2025), 103049. doi:10.1016/j.aei.2024.103049

[10] Lianhong Ding, Na Ding, and Peng Shi. 2026. TempRAA: temporal relation-aware alignment for enhancing LLMs reasoning in time-sensitive knowledge graph question answering. *Journal of Intelligent Information Systems* (2026). doi:10.1007/s10844-026-01027-w

[11] Gabe Fierro, Marco Pritoni, Moustafa Abdelbaky, Daniel Lengyel, John Leyden, et al. 2019. Mortar: An Open Testbed for Portable Building Analytics. *ACM Trans. Sen. Netw.* 16, 1, Article 7 (Dec. 2019), 31 pages. doi:10.1145/3366375

[12] Jessica Granderson, Guanjing Lin, Yimin Chen, Armando Casillas, et al. 2022. LBNL Fault Detection and Diagnostics Datasets. Open Energy Data Initiative (OEDI), Lawrence Berkeley National Laboratory, https://doi.org/10.25984/1881324. doi:10.25984/1881324 Accessed: 2025-12-08.

[13] Haoyu Han, Yu Wang, Harry Shomer, Kai Guo, et al. 2025. Retrieval-Augmented Generation with Graphs (GraphRAG). arXiv:2501.00309 [cs.IR] https://arxiv.org/abs/2501.00309

[14] Steve Harris and Andy Seaborne. 2013. *SPARQL 1.1 Query Language.* W3C Recommendation. W3C. https://www.w3.org/TR/sparql11-query/

[15] Shaoxiong Ji, Shirui Pan, Erik Cambria, Pekka Marttinen, and Philip S. Yu. 2022. A Survey on Knowledge Graphs: Representation, Acquisition, and Applications. *IEEE Transactions on Neural Networks and Learning Systems* 33, 2 (Feb. 2022), 494–514. doi:10.1109/tnnls.2021.3070843

[16] Erkan Karabulut, Salvatore F. Pileggi, Paul Groth, and Victoria Degeler. 2024. Ontologies in digital twins: A systematic literature review. *Future Generation Computer Systems* 153 (2024), 442–456. doi:10.1016/j.future.2023.12.013

[17] Yunshi Lan, Gaole He, Jinhao Jiang, Jing Jiang, et al. 2021. A Survey on Complex Knowledge Base Question Answering: Methods, Challenges and Solutions. arXiv:2105.11644 [cs.CL] https://arxiv.org/abs/2105.11644

[18] Nelson F. Liu, Kevin Lin, John Hewitt, Ashwin Paranjape, et al. 2023. Lost in the Middle: How Language Models Use Long Contexts. arXiv:2307.03172 [cs.CL] https://arxiv.org/abs/2307.03172

[19] Shicheng Liu, Sina Semnani, Harold Triedman, Jialiang Xu, et al. 2024. SPINACH: SPARQL-Based Information Navigation for Challenging Real-World Questions. In *Findings of the Association for Computational Linguistics: EMNLP 2024*, Yaser Al-Onaizan, Mohit Bansal, and Yun-Nung Chen (Eds.). Association for Computational Linguistics, Miami, Florida, USA, 15977–16001. doi:10.18653/v1/2024.findings-emnlp.938

[20] Ozan Baris Mulayim, Avia Anwar, Umut Mete Saka, Lazlo Paul, et al. 2025. BuildingQA: A Benchmark for Natural Language Question Answering over Building Knowledge Graphs. In *Proceedings of the 12th ACM International Conference on Systems for Energy-Efficient Buildings, Cities, and Transportation* (Colorado School of Mines, Golden, CO, USA) *(BuildSys '25)*. Association for Computing Machinery, New York, NY, USA, 65–75. doi:10.1145/3736425.3770097

[21] Ozan Baris Mulayim, Lazlo Paul, Marco Pritoni, Anand Krishnan Prakash, et al. 2024. Large Language Models for the Creation and Use of Semantic Ontologies in Buildings: Requirements and Challenges. In *Proceedings of the 11th ACM International Conference on Systems for Energy-Efficient Buildings, Cities, and Transportation* (Hangzhou, China) *(BuildSys '24)*. Association for Computing Machinery, New York, NY, USA, 312–317. doi:10.1145/3671127.3698792

[22] Marco Pritoni, Drew Paine, Gabriel Fierro, Cory Mosiman, et al. 2021. Metadata Schemas and Ontologies for Building Energy Applications: A Critical Review and Use Case Analysis. *Energies* 14, 7 (2021). doi:10.3390/en14072024

[23] Project Haystack. 2026. Project Haystack - An Open Source initiative to streamline working with IoT Data. https://www.project-haystack.org/. Accessed: 2026-05-03.

[24] Bhaskarjit Sarmah, Dhagash Mehta, Benika Hall, Rohan Rao, et al. 2024. HybridRAG: Integrating Knowledge Graphs and Vector Retrieval Augmented Generation for Efficient Information Extraction. In *Proceedings of the 5th ACM International Conference on AI in Finance* (Brooklyn, NY, USA) *(ICAIF '24)*. Association for Computing Machinery, New York, NY, USA, 608–616. doi:10.1145/3677052.3698671

[25] Priyanka Sen, Armin Oliya, and Amir Saffari. 2021. Expanding End-to-End Question Answering on Differentiable Knowledge Graphs with Intersection. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, Marie-Francine Moens, Xuanjing Huang, Lucia Specia, and Scott Wen-tau Yih (Eds.). Association for Computational Linguistics, Online and Punta Cana, Dominican Republic, 8805–8812. doi:10.18653/v1/2021.emnlp-main.694

[26] Haitian Sun, Bhuwan Dhingra, Manzil Zaheer, Kathryn Mazaitis, et al. 2018. Open Domain Question Answering Using Early Fusion of Knowledge Bases and Text. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, Ellen Riloff, David Chiang, Julia Hockenmaier, and Jun'ichi Tsujii (Eds.). Association for Computational Linguistics, Brussels, Belgium, 4231–4242. doi:10.18653/v1/D18-1455

[27] Jiashuo Sun, Chengjin Xu, Lumingyuan Tang, Saizhuo Wang, et al. 2024. Think-on-Graph: Deep and Responsible Reasoning of Large Language Model on Knowledge Graph. arXiv:2307.07697 [cs.CL] https://arxiv.org/abs/2307.07697

[28] Alon Talmor and Jonathan Berant. 2018. The Web as a Knowledge-Base for Answering Complex Questions. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*, Marilyn Walker, Heng Ji, and Amanda Stent (Eds.). Association for Computational Linguistics, New Orleans, Louisiana, 641–651. doi:10.18653/v1/N18-1059

[29] Wikidata. 2026. *Wikidata:SPARQL tutorial.* https://www.wikidata.org/wiki/Wikidata:SPARQL_tutorial Wikidata Help Page.

[30] Guanming Xiong, Junwei Bao, and Wen Zhao. 2024. Interactive-KBQA: Multi-Turn Interactions for Knowledge Base Question Answering with Large Language Models. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Association for Computational Linguistics, 10561–10582. doi:10.18653/v1/2024.acl-long.569

[31] Silei Xu, Shicheng Liu, Theo Culhane, et al. 2023. Fine-tuned LLMs Know More, Hallucinate Less with Few-Shot Sequence-to-Sequence Semantic Parsing over Wikidata. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, Houda Bouamor, Juan Pino, and Kalika Bali (Eds.). Association for Computational Linguistics, Singapore, 5778–5791. doi:10.18653/v1/2023.emnlp-main.353

[32] Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, et al. 2023. ReAct: Synergizing Reasoning and Acting in Language Models. arXiv:2210.03629 [cs.CL] https://arxiv.org/abs/2210.03629

[33] Xi Ye, Semih Yavuz, Kazuma Hashimoto, Yingbo Zhou, and Caiming Xiong. 2022. RNG-KBQA: Generation Augmented Iterative Ranking for Knowledge Base Question Answering. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, Smaranda Muresan, Preslav Nakov, and Aline Villavicencio (Eds.). Association for Computational Linguistics, Dublin, Ireland, 6032–6043. doi:10.18653/v1/2022.acl-long.417

[34] Wen-tau Yih, Ming-Wei Chang, Xiaodong He, and Jianfeng Gao. 2015. Semantic Parsing via Staged Query Graph Generation: Question Answering with Knowledge Base. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, Chengqing Zong and Michael Strube (Eds.). Association for Computational Linguistics, Beijing, China, 1321–1331. doi:10.3115/v1/P15-1128

[35] Wen-tau Yih, Matthew Richardson, Chris Meek, Ming-Wei Chang, and Jina Suh. 2016. The Value of Semantic Parse Labeling for Knowledge Base Question Answering. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, Katrin Erk and Noah A. Smith (Eds.). Association for Computational Linguistics, Berlin, Germany, 201–206. doi:10.18653/v1/P16-2033

[36] Yusen Zhang, Ruoxi Sun, Yanfei Chen, Tomas Pfister, et al. 2024. Chain of Agents: Large Language Models Collaborating on Long-Context Tasks. In *Advances in Neural Information Processing Systems*, A. Globerson, L. Mackey, D. Belgrave, A. Fan, U. Paquet, J. Tomczak, and C. Zhang (Eds.), Vol. 37. Curran Associates, Inc., 132208–132237. doi:10.52202/079017-4202