World Wide Web (2026) 29:17
https://doi.org/10.1007/s11280-026-01403-0

RESEARCH

OntoSage: Intelligent Human-Building Smartbot for Semantic Smart Building Question Answering

Suhas Devmane[1] · Omer Rana[1] · Charith Perera[1]

Received: 30 April 2025 / Revised: 13 October 2025 / Accepted: 6 January 2026

Abstract
Smart buildings remain heterogeneous across sensing infrastructure, metadata quality, legacy protocols, and analytics requirements, hindering reusable human–building natural language interfaces. We present **OntoSage**, a modular framework for ontologically grounded question answering (QA) and fulfillment of analytic intents over smart building data. The framework (i) leverages Brick Schema-based RDF model with reasoning capabilities, (ii) translates natural language (NL) questions into executable SPARQL via a fine-tuned seq2seq model (T5-Base), and (iii) orchestrates portable analytics microservices that operate on time-series sensor data referenced through ontology-linked UUIDs. A summarization component (open-weights Mistral-7B, zero-shot) converts structured SPARQL/SQL/analytic outputs into concise stakeholder-aware responses without requiring task-specific fine-tuning. We categorize QA complexity into four reasoning classes and report component-level execution metrics supporting these categories. To address portability, we formalize a lightweight adaptation workflow (ontology ingestion→entity enrichment for NLU→NL2SPARQL validity checks→analytics binding) designed to minimize per-building retraining. Reproducibility is enabled through public source code, synthetic and ontology-derived datasets, Docker/Compose service descriptors, and documented supporting scripts "(https://github.com/suhasdevmane/OntoBot)". The developers' documentation is publicly accessible "(https://ontosage-docs.github.io)".

Keywords Human-Building Interaction (HBI) · Internet of Things (IoT) · Smart Buildings · Semantic Web · Ontologies · Large Language Models(LLMs)

✉ Suhas Devmane
DevmaneSP1@cardiff.ac.uk

Omer Rana
RanaOF@cardiff.ac.uk

Charith Perera
PereraC@cardiff.ac.uk

[1] School of Computer Science and Informatic, Cardiff University, Cathays, Cardiff CF24 4AG, Wales, UK

Published online: 20 January 2026

1 Introduction

Human–Building Interaction (HBI) studies how occupants, cyber-physical infrastructure, and data services co-evolve in intelligent built environments [1, 2]. Contemporary smart buildings integrate heterogeneous sensing (environmental, HVAC, air quality, etc.), varied middleware, inconsistent metadata curation, and siloed Building Management Systems (BMS). This fragmentation impedes the development of universal, reusable natural language (NL) interfaces that would allow stakeholders (facility managers, sustainability officers, maintenance staff, visitors, etc.) to interrogate both semantic knowledge (ontologies) and live/historical time-series data. Commodity voice/NL platforms (Alexa, Google Home, HomeKit, SmartThings) provide intent execution abstractions but (i) lack explicit semantic grounding in standard building ontologies, (ii) cannot express multi-hop or ontology+timeseries fused analytical queries, and (iii) require opaque, proprietary adaptation. Generic large language models (LLMs) exhibit fluent dialogue yet hallucinate unseen device semantics and offer no built-in alignment to Brick-structured relational queries.

Three persistent gaps motivate this work: (1) *Reasoning coverage*- existing prototypes emphasize direct entity lookups rather than aggregated, multi-relation, or temporal/analytic intents; (2) *Portability*-per-building re-engineering of training data and pipeline configuration limits scalable deployment; (3) *Reproducibility and extendability*- NL→SPARQL demonstrations omit dataset construction details, versioned artifacts, or adaptation workflows for future advancements. Moreover, no domain-specific LLMs for the smart buildings domain are available for NL-to-SPARQL translation in QA.

We present **OntoSage**, a modular ontology-based framework for QA and analytics orchestration that converts natural language questions into SPARQL. It links results to time-series analytics microservices and summarizes outcomes for users. The Brick Schema [3] serves as a shared semantic foundation, allowing each asset to expose a UUID that connects RDF descriptors to structured sensor telemetry. A fine-tuned T5-Base seq2seq model performs NL→SPARQL translation; a domain-aware Rasa Natural Language Understanding (NLU) layer mediates entity normalization and slot completion; portable analytics microservices receive normalized JSON payloads; an open-weights Mistral-7B model (zero-shot) produces human-readable rationales. This paper makes the following contributions to HBI and semantic smart building QA:

1. Unified Reasoning Taxonomy: We define four NL reasoning classes (single-hop factual, multi-hop relational, aggregation/temporal, ontology+timeseries fusion) and align component evaluation metrics to each.
2. Portable Adaptation Workflow: A minimal four-stage process (ontology ingestion→ entity enrichment→NL to SPARQL conformance validation→analytics binding) enabling reuse across buildings with reduced retraining.
3. Open Reproducibility Stack: Public release of code, synthetic + ontology-derived NL–SPARQL pairs, Docker/Compose deployment descriptors, training/evaluation scripts, and developers' documentation for extending goals.
4. Microservice Analytics Layer: Extensible, decoupled analytic endpoints (anomaly detection, comfort indices, air quality aggregation, correlation, forecasting, etc.) callable directly from SPARQL-resolved entity UUIDs.

5. Empirical Component Evaluation: A quantitative assessment of NLU entity extraction, SPARQL generation, analytics accuracy, and summarization quality based on a real-world testbed.
6. Limitations and Roadmap Articulation: A transparent analysis of challenges in multi-entity disambiguation, long-tail intent coverage, and dataset breadth constraints.

We address the following research questions:

1. RQ1 (System): What system components and architecture are required to support human-building conversations with smart buildings, enabling multi-persona question answering at scale?
2. RQ2 (Applications): How can a system be constructed for heterogeneous building environments so that non-expert users can contribute to mutual human-building benefits?
3. RQ3 (Analytics): How can portable analytic applications be implemented for the built environment, covering installed systems to answer analytical questions using data collected in smart buildings?

The primary objectives of this research are:

- Develop a framework that enables diverse users, from guests to experts, to interact with smart buildings in natural language, providing a range of analytical and operational insights of the built environment to enhance building health and sustainability.
- Fine-tune LLMs and develop adaptable algorithms as microservices that allow users to leverage real-time and historical building data for multiple objectives that can be deployed in heterogeneous built environments.
- Establish a standardized, conversational AI-driven approach with NLU and LLMs to effortlessly incorporate new building types and components into the framework, minimizing reconfiguration while ensuring robust, domain-specific responses.

We introduce a scalable framework validated through a multi-month deployment in a real-world building testbed with diverse environmental sensors. Its modular design allows for future analytical extensions and replication across buildings. **Paper organization:** Section 2 reviews related work in NL-to-SPARQL, smart building ontologies, and conversational agents. Section 3 outlines the system architecture and model design, including the NLU pipeline, NL-to-SPARQL translation, and analytics orchestration. Section 4 describes the experimental setup, covering testbed development, NLU training, and T5 model training. Section 5 presents component evaluations, baseline comparisons, reasoning class analysis, and cross-building portability. Section 6 outlines analytics applications. Section 7 states limitations and future enhancements. Section 8 examines implications, and Section 9 concludes. Extended chatbot QA examples can be found in Appendix A.

2 Related Work

The literature relevant to OntoSage spans (i) NL→SPARQL translation, (ii) semantic/ontology modeling for buildings, and (iii) conversational agents for HBI.

2.1 NL to SPARQL Query Generation

Early template and rule-driven pipelines [4, 5] have largely given way to encoder-decoder Transformers (e.g., BART, T5) and hybrid entity-linking augmented models [6–11]. Recent work leverages large language models for broader generalization and compositionality [12–16], yet challenges persist for multi-hop joins, temporal aggregation, and faithful entity grounding-core reasoning classes we explicitly benchmark. Translating natural language queries into structured SPARQL queries for knowledge graph interrogation has been a cornerstone of recent research. Early efforts, such as [4], applied NLP techniques to datasets like Stanford SQuAD, enabling natural language query processing and laying the foundation for subsequent advancements. However, these approaches often relied on template-based systems, which required manual effort to construct domain-specific query templates and struggled to adapt to new knowledge graphs [5]. To overcome these limitations, recent studies have explored end-to-end deep learning frameworks and hybrid systems. For instance, [6] demonstrated the efficacy of fine-tuning pre-trained encoder-decoder models, such as BART [17] and T5 [7], to generate SPARQL queries. This approach handles unseen entities by translating entity IDS into text labels. Similarly, [18] evaluated pre-trained versus non-pre-trained models, demonstrating that techniques such as question annotation and copy mechanisms improve query generation accuracy. Transformer-based models have further advanced this domain. Fine-tuned models, such as T5 and SPBERT [8, 9], achieve superior performance on benchmark datasets compared to task-specific models. Hybrid approaches combining neural machine translation (NMT) with entity linking have also gained traction. Moreover,[10] and [11] integrated NMT with dedicated entity linking to bridge the gap between ambiguous natural language and structured SPARQL queries, a strategy reinforced by [19] and [20]. The rise of large language models (LLMs) has accelerated progress, with [12–15], and [16] leveraging LLMs to encode linguistic features and embed knowledge graphs, enabling executable SPARQL queries for complex domains.

2.2 Ontologies for Smart Buildings

The evolution of smart building ontologies reflects a shift from narrow, energy-focused schemas to robust, interoperable metadata standards. Early frameworks such as Green Building XML (gbXML) and the Building Energy Data Exchange Specification (BEDES) [21–23] laid the groundwork for energy modeling and data exchange, but their limited expressivity constrained broader interoperability. Subsequent efforts addressed these limitations by introducing standardized vocabularies and extensible frameworks. Notable examples include Project Haystack, which enhanced semantic modeling for building management systems, and Industry Foundation Classes (IFC), which supported design-phase interoperability in architecture and engineering domains [24, 25]. The emergence of IoT-centric ontologies, such as the Smart Appliances REFerence Ontology (SAREF), provided structured representations for smart appliances, thereby advancing device interoperability and application integration [26–29]. However, these schema often lacked adaptability across diverse building types. More flexible approaches were introduced through the Building Topology Ontology (BOT) and Semantic Sensor Network (SSN)/SOSA standards, both of which leveraged RDF-based models to address operational and sensing requirements while remaining specialized to particular domains [30–32].

The development of the Brick ontology represents a significant milestone in this progression. Brick offers an open-source, RDF-based schema that prioritizes completeness, extensibility, and uniformity of metadata for both physical and virtual assets in smart buildings [33, 34]. Unlike traditional OWL-based ontologies constrained by Description Logic(DL), Brick utilizes RDF and RDFS semantics, facilitating more nuanced modeling of sensor networks and diverse building subsystems. Despite these advances, ontologies still face challenges in automated cross-building alignment and the seamless coupling of semantic descriptors with live analytic microservices. Developing building-agnostic, portable metadata frameworks that integrate with conversational AI and tap into underutilized IoT sensor data (often stored in structural databases) remains an open research frontier. Our work addresses these gaps by introducing a UUID-based linkage and portability workflow, which enhances cross-building interoperability and supports real-time analytic integration.

2.3 Conversational Agents in HBI

Conversational agents, also known as chatbots, have become pivotal interfaces for human-building interaction (HBI), enabling natural language interactions with intelligent environments. Traditional chatbots, relying on pattern-matching rules, offered limited conversational capabilities [35], which highlighted the need to formalize both rational (reasoning and NLP) and intuitive (semantic) components for human-like dialogue. Recent NLP advances, particularly transformer-based models and LLMs like GPT, have transformed chatbot development by enabling flexible, context-aware dialogue through prompting [36]. Frameworks like Rasa [37] and Dialogflow, which are widely used for intent recognition and entity extraction, have been instrumental. For instance, [38] integrated Rasa with SPARQL generation modules to address query ambiguity, while [39] combined Rasa-based entity extraction with custom SPARQL modules for improved question-answering accuracy. Linguistic rule-based systems, such as those employing syntactic ambiguity resolution [40], have further enhanced intent detection.

AI-based methods, including Transformers and reinforcement learning [36], dominate chatbot development for IoT, supporting applications such as patient monitoring [41], human activity recognition [42, 43], security [44], and energy efficiency [45]. Ontology-based chatbots have demonstrated versatility across various domains, including tutoring [46], e-commerce [47], and healthcare [48], by leveraging ontologies to enhance response generation. Domain-specific systems, such as [49] for Korean query answering and KBot [50] for smart home interactions, merge linked data with machine learning (ML) for interactive question answering. Recent HBI studies emphasize user psychology and multimodal interactions. For example, [51] found audio output modalities influence perceptions of sensitive information retrieval, while [52] highlighted challenges in privacy, multi-user experiences, and design considerations [53–58]. Systems like TAO [59], combining ontological and unsupervised clustering approaches, can infer rich contexts from daily activities, underscoring the potential of integrating semantic technologies with conversational AI for adaptive, user-friendly smart building interfaces.

3 System Architecture and Design

This section presents the OntoSage framework's architecture, encompassing foundational requirements Section 3.1, core system components Section 3.2, model architecture including NLU and NL-to-SPARQL translation Section 3.3, post-processing and data retrieval Section 3.4, analytics pipeline Section 3.5, and summarization Section 3.6. These elements collectively enable multi-building semantic question answering over diverse building infrastructures. Sequence-to-sequence LLMs have been used in our previous work [60], and the optimization of SPARQL for domain-specific queries points to promising directions for the future.

3.1 Architectural Requirements

Exploring current technologies and open-source components to facilitate the human-building conversation for multiple goals, we found that we need a standard description logic and a language to communicate domain knowledge, as well as a pipeline consisting of microservices that will perform their specific tasks when supplied with the right entities. Figure 1 illustrates the system overview with its main components: Graphical User Interfaces (GUI) and backend processes. The GUI connects to different services, enabling user communication with smart buildings, and consists of interconnected services necessary to complete the conversation. It provides natural language question answering (NLQA) in smart building environments by integrating semantic ontologies, large language models (LLMs), a modular backend, and a conversational AI interface.

The following is a list of system components required to assemble a meaningful QA approach addressing the first research objective (RQ 1).

- Semantic Description (BrickSchema Ontology): Utilises the BrickSchema terminology to provide a comprehensive semantic description of physical, logical, and virtual assets, capturing their relationships within the smart building ecosystem.
- Ontology Reasoning (SHACL, OWLRL, VBIS): Employs SHACL, OWLRL, and VBIS to validate RDF data against predefined shapes, ensuring compliance with constraints and enriching the ontology with domain-specific knowledge.
- Ontology Query Language (SPARQL): Leverages SPARQL to efficiently retrieve information from RDF databases, enabling precise querying within the context of Brick-Schema terminology and the Building RDF model.
- RDF Triple Store (Apache Jena Fuseki): We chose Apache Jena Fuseki from other open-source SPARQL servers to host the smart building model. It can run as a standalone

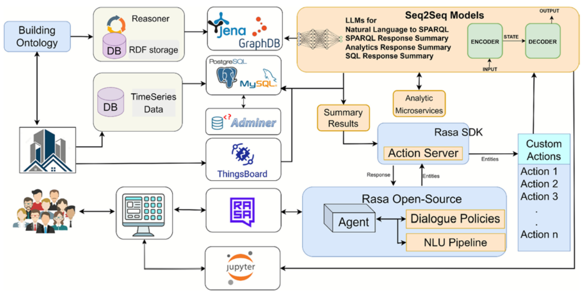

Fig. 1 Overview of system architecture

server or be embedded in an application. Fuseki provides the SPARQL protocol and SPARQL Graph Store protocol for querying and updating data. It is integrated with TDB to provide robust, transactional, persistent storage, and related reasoners can be applied to the RDF models to extend the RDF Model Terminologies by performing logical inferences.

- Conversational Chatbot (RaSa Open Source): Integrates RaSa Open Source to enable interactions with the smart building system, extracting entities and offering fallback mechanisms to gather additional user information using NLU.
- Timeseries Database (Postgresql/Mysql, etc.): Employs a structured database to store large volumes of sensor/device data from the smart building. All timeseries data is uniquely identified in the building's RDF Model for efficient retrieval and processing.
- IoT Platform (ThingsBoard): Utilizes ThingsBoard to collect, process, visualize, and analyze data from IoT devices, seamlessly integrated with PostgreSQL and PgAdmin for robust data management and administration.
- Natural Language to SPARQL Translation (T5 (t5-base)): Applies the T5 (t5-base) model to translate natural language questions into SPARQL queries, leveraging natural language understanding entities for accuracy.
- SPARQL Summarisation (Mistral 7B): Open source Seq2Seq models are explored, and Mistral 7B LLMs are used to generate concise and meaningful summaries of responses from the knowledge base, database, and analytic microservices to enhance user comprehension.
- Training Datasets (Custom): Employs custom datasets to train Seq2Seq LLMs for natural language to SPARQL translation, embedding domain-specific knowledge for improved performance.
- Analytics Operation on the data (Microservices): A server hosting analytic applications to perform data analytics. Rasa's NLU is trained on brick schema terminology to extract Entities, and the related timeseries data is used for analytics.

3.2 Core Components and Data Flow

3.2.1 Core Components

Imagine a smart built environment, equipped with multiple systems that comprise a sensor/device network deployed across various zones, as shown in Figure 2. This installed built environment is modeled with a formal BrickSchema terminology, a widely adopted schema chosen for its standardized vocabulary and extensible structure. The ontology, represented in Turtle (TTL) format, is parsed into RDF triples and stored in a triple store (SPARQL server) with reasoning capabilities, such as GraphDB or Apache Jena Fuseki. On the other hand, all such devices' data is stored in structured databases such as MySQL or PostgreSQL. Each sensor/device with its unique time series reference ID is added to the RDF Model of the building, with spatial relationships and properties.

On the User side, the framework leverages 'Rasa Open Source' for dialogue management, intent recognition, and entity extraction. Rasa's NLU pipeline is configured using YAML files that define intents, entities, lookup tables, regex patterns, synonyms, and dialogue rules. A key design goal is to identify smart building-specific entities, such as sensor names, dates, and locations, and analyze the type context during conversations. Ontology-

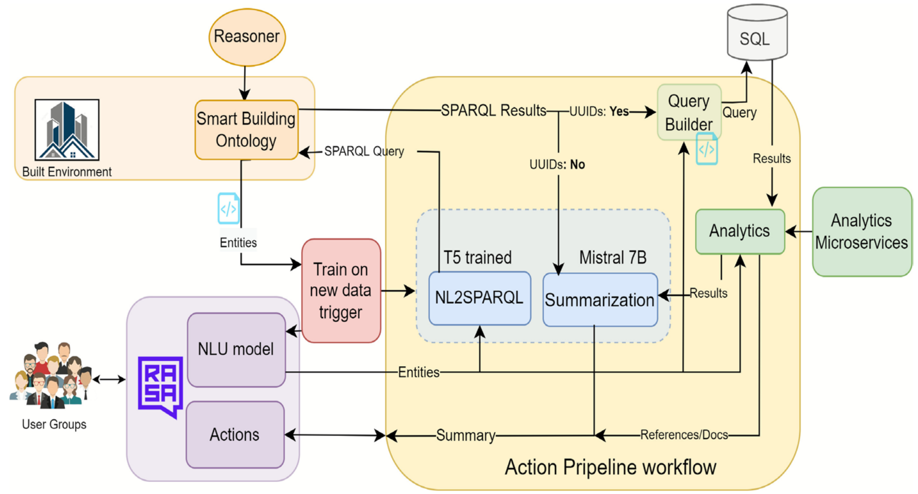

Fig. 2 Machine learning model and custom actions flow

derived entities from the RDF graph are embedded directly into Rasa's training data to improve entity recognition. Rasa includes a fallback policy to handle ambiguous inputs, redirecting the user to a data-gathering phase to collect all required slots. Once all necessary fields are filled, the 'Rasa SDK Action Server' triggers corresponding custom actions, defined in Python, which execute database queries, API calls, or analytical routines.

To enable intelligent and context-aware interactions with this structured data, we fine-tuned **T5 models** to generate SPARQL queries from natural language questions. These dynamically constructed queries interact with the locally chosen SPARQL server, retrieving precise results that accurately reflect the user's intent. Based on the SPARQL response, collected UUIDS are used in parallel to retrieve time-series data from the database through a standard SQL query template (for chosen database) that includes entity placeholders. The framework employs a dual-model architecture to provide high-quality responses. A fine-tuned T5 model handles query generation, while a **Mistral 7B model** summarises responses, whether from SPARQL, or SQL and analytic microservices. This combination ensures both accuracy in retrieval and clarity in output. The summarisation endpoint is effective at describing responses and providing interpretable insights for end-users.

ThingsBoard, an open-source IoT platform, enables device visualization, data flow management, real-time monitoring, alerting, and administrative control within local networks. It supports multiple sensor protocols and stores data in structured databases for efficient access. Tools such as Adminer and PgAdmin assist with database management and debugging, while Jupyter Notebooks integrate easily for rapid prototyping. Extensible analytic microservices operate independently to process complex sensor datasets, with results from SPARQL endpoints, databases, and analytics routed through an action server for clear summaries. By combining BrickSchema, SPARQL, Rasa, structured databases, analytic microservices, and advanced language models, the framework delivers a scalable and semantically rich NLQA solution for smart buildings, promoting intuitive interaction, structured querying, and sustainable operations.

3.2.2 Data Flow

User interaction begins with a chatbot interface, where users pose queries related to the environment, sensor data, or building-specific analytics. These inputs are first processed by an NLU pipeline powered by Rasa Open Source. The NLU model is trained using data derived from the smart building ontology (with TBox terms from Brick V1.4), enabling it to accurately classify intents and extract key entities such as sensor types(names), locations, dates, and analytic terms. To facilitate this training, we implemented a dedicated function that ingests the ontology, extracts relevant entity labels (sensors-uuids mappings), and incorporates them into the training data as synonyms, lookup tables, and regex patterns. The smart building ontology model must use TBox (Terminological Box), i.e., the vocabulary, structure, and relationships between concepts (classes and properties) of BrickSchema (V1.4) terminology as a basis. When the NLU component identifies an intent, it forwards the structured output (intent and entities) to the custom action module. Even if the intent is classified as out of scope, the framework is designed to route the input to a fallback action for robustness testing. Within the custom action script, the first key component is an implementation of a fine-tuned T5 model trained to perform NL2SPARQL translation. This model receives the user's prompt, enhanced with extracted entities, and generates a SPARQL query representing the user's question. The generated SPARQL query is executed against the smart building ontology, which is hosted in an SPARQL server. Ontological reasoning is facilitated via a reasoner (currently supports rdfs, owlrl, vbis, shacl reasoning, implemented in a Jupyter environment) that supports relationship inference and schema validation. The resulting RDF data is returned in JSON format for further processing. A decision point in the workflow checks whether the SPARQL result contains unique identifiers (UUIDS), which indicate links to time-series sensor data stored in relational databases. If such UUIDS are present, the query is passed to a query builder module, which populates a predefined standard SQL query template (unique for the employed database type) with appropriate entity values such as sensor UUIDs, and date range. The constructed SQL query is sent to a chosen structural database endpoint, which returns the result as JSON. At the same time, when UUIDs are present in the SPARQL response, scripts trigger a service call to decide whether to need to perform analytics and which analytics application to perform with a decider service Section 3.5.1.

The response (whether derived from SPARQL or SQL) is then routed to the analytics layer Section 3.5. This component may invoke external analytics microservices to perform advanced computations such as trend analysis, aggregation, or anomaly detection. These microservices can also produce external references, graphical summaries, and additional JSON outputs to enhance interpretability. Once analytics are complete, the results are passed to a summarisation module powered by the Mistral 7B model. This model can generate coherent and context-aware summaries, transforming raw RDF triples or structured database responses into fluent, user-friendly, stakeholder profile-based explanations. In cases where the SPARQL query does not yield time-series data, the output is immediately forwarded to the summarisation stage without invoking the SQL and analytics layer.

The final summarised response is sent back to the chatbot interface and presented to the user. For traceability and debugging, all query responses and analytics results are also stored/shared in JSON/png format, which can be accessed via UI and analyzed deeply through an integrated Jupyter Notebook interface. The system includes a training trigger mechanism that detects newly added building data or changes in ontology structure to ensure adaptability. This mechanism periodically re-trains the NL2SPARQL model, ensuring the pipeline remains aligned with the evolving smart building environment. The framework enables end-users, regardless of technical expertise, to query complex building systems and receive actionable insights in natural language.

3.3 Model Architecture

3.3.1 NLU Pipeline

At the front end, a Rasa-powered conversational agent captures natural language queries and converts them into structured requests for downstream processing. The unique NLU model for a single built environment within Rasa is trained to detect key intents and extract relevant entities such as sensor types, time ranges, and locations from user queries using a custom pipeline specified in the `config.yml` file. For instance, when a user inquires, "Which rooms currently have a temperature above the recommended setpoint (e.g., 24 °C)?", the NLU component identifies the intent as `query_temperature_data` and extracts entities such as `temperature_sensor`, `Room_x`, `start_date`, `end_date`, and `setpoint`.

As shown in the Figure 3, the Rasa NLU pipeline processes user queries, such as "Which rooms currently have a temperature above the recommended setpoint (e.g., 24°C)?",

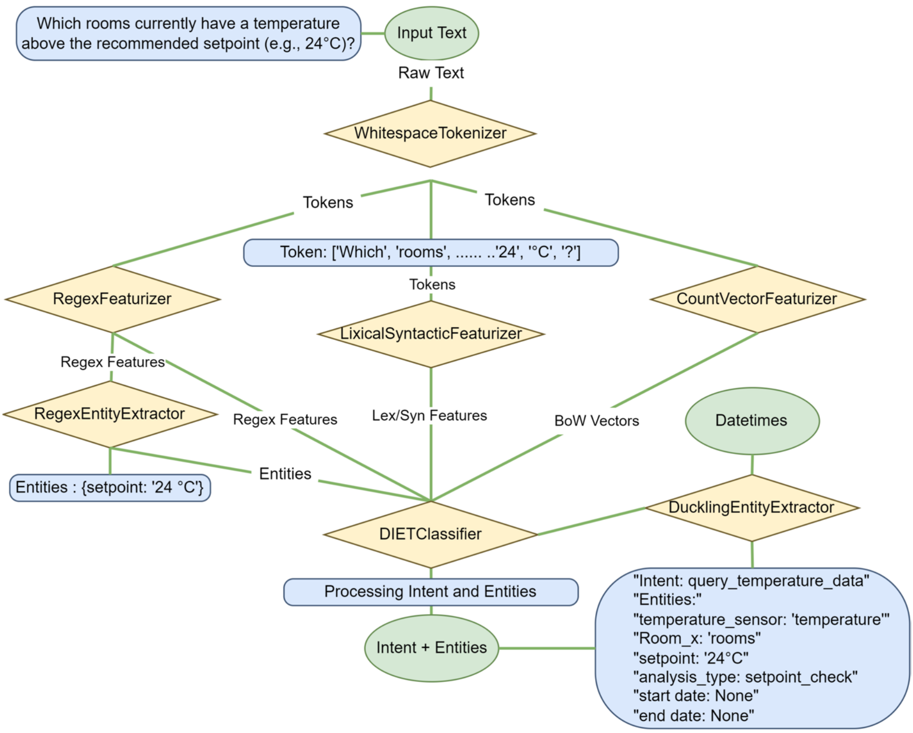

Fig. 3 Rasa NLU pipeline for intent classification and entity extraction

through a series of components to classify intents and extract entities, followed by dialogue management to trigger appropriate actions. Initially, the `WhitespaceTokenizer` splits the input text into tokens based on whitespace, producing a list like ["Which", "rooms", , "24°C)?"]. The `RegexFeaturizer` then applies regular expression patterns defined in the training data, such as `\d+°C` for temperatures or `setpoint` for specific terms, to generate features for entities and intents. Next, the `RegexEntityExtractor` identifies entities directly using these patterns, yielding a list as shown in the following box:

```
[
  {"entity": "temperature", "value": "20", "start": 45, "end": 47},
  {"entity": "setpoint",    "value": "25", "start": 33, "end": 35}
]
```

The `LexicalSyntacticFeaturizer` enriches tokens with lexical features (e.g., part-of-speech tags like "rooms: NOUN") and syntactic features (e.g., dependency parsing), providing linguistic context to aid downstream components. The `CountVectorsFeaturizer` converts the text into a numerical bag-of-words vector, enabling intent classification. The `DIETClassifier`, a transformer-based model, leverages features from all prior components to jointly classify the intent (e.g., `action_question_to_brickbot` with a confidence score) and refine entities (e.g., `temperature_sensor`, `Room_x`, `setpoint`). Alongside, the Duckling server is running as a microservice to extract the dates using the Rasa Duckling image. Dialogue management is handled by the `MemoizationPolicy`, which matches the conversation state (intent + entities) to known training stories to predict action; the `RulePolicy`, which applies predefined rules to execute actions (e.g., triggering `action_question_to_brickbot` for the `ActionQuestionToBrickbot` class); and the `TEDPolicy`, a transformer-based model that predicts actions based on dialogue history, intent, and entities, excelling in unseen scenarios.

3.3.2 NL to SPARQL Translation

Our previous study [60] identified the T5-base model as the most suitable for this task. We trained the model using natural language questions paired with formatted SPARQL queries, including necessary prefixes. The training dataset contains over 120,000 example pairs in a multi-variant NL question format, enabling the model to generate SPARQL queries from user input. Since most smart building sensor devices generate time-series data stored in structured databases (e.g., MySQL, PostgreSQL), directly adding large volumes of time-series data to the RDF model is Inappropriate. Instead, referencing the time-series data's location in the RDF model enables a more efficient solution for linking the structured databases with the RDF model. Each sensor or device is assigned a unique UUID, as defined by BrickSchema TBox terminology. When a user submits a query, the system extracts entities, identifies the intent, and retrieves relevant time-series IDs. Once the entities are identified, the NL question and contextual information are sent to the model, which generates the corresponding SPARQL query, even if TBox information is missing. For example, in response to the question, "Tell me the failure trends of the exhaust air flow sensor in the Maintenance Room," the NLU pipeline extracts entities such as `"analysis_type"`: `"failure trends"`, `"sensor_type":"exhaust air flow sensor"`, and "location": "maintenance Room." The T5 model, using these entities along with BrickSchema T-box terms (e.g., brick:hasLocation) and A-box instances (e.g., bldg:Maintenance_Room), generates the following SPARQL query:

```
{
    "question": "Tell me the failure trends of the exhaust air flow
        ↪ sensor in the Maintenance Room.",
    "entity": "bldg:Maintenance_Room \n
     brick:Exhaust_Air_Flow_Sensor",
    "sparql": "
        SELECT ?sensor ?timeseriesId ?storedAt
        WHERE {
            ?sensor a brick:Exhaust_Air_Flow_Sensor ;
                    brick:hasLocation bldg:Maintenance_Room ;
                    ref:hasExternalReference ?ref .
            ?ref a ref:TimeseriesReference ;
                 ref:hasTimeseriesId ?timeseriesId ;
                 ref:storedAt ?storedAt .
        }"
}
```

This query retrieves the sensor, timeseries ID, and data storage location, facilitating efficient data retrieval from structured databases for next steps.

3.4 Post-Processing and Data Retrieval

After executing the SPARQL query against the SPARQL endpoint, the system processes the results to extract time-series IDs, which are used to retrieve sensor data from structured databases. If no IDs are retrieved, a fallback mechanism employing the open-source Mistral LLMs generates summarized responses. The retrieved data is formatted into a standardized JSON structure, enhancing compatibility with subsequent analysis and summarization stages. The post-processing workflow includes time-series ID extraction, database querying, result formatting, and data retrieval. The SPARQL query results are parsed to identify time series IDs corresponding to building sensors/devices (ABox individuals). These IDs, aligned with BrickSchema UUIDs, serve as references to time series data stored in structured databases (e.g., MySQL, PostgreSQL). If IDs are absent, the system invokes the Mistral LLMs to generate a summarized response or prompt the user for additional input via fallback functions, ensuring robust handling of incomplete data.

The extracted time-series IDs are used to populate placeholders in predefined SQL queries. If placeholders cannot be filled due to missing entities, fallback mechanisms (e.g., user prompts or default values) ensure query completeness. The SQL query retrieves time-series data, including timestamps and sensor readings, from the specified database table. This approach leverages the scalability of structured databases while maintaining semantic links to the RDF model, thereby ensuring seamless integration and consistency. The SQL query results are converted into a standardized JSON format, with sensor names as top-level keys, replacing UUIDs to provide contextual clarity. This structure, shown in the fetch_sensor_data function below, organizes time-series data by sensor, facilitating integration with downstream analysis and summarisation pipelines.

```
def fetch_sensor_data(timeseries_sensors_map, start_time, end_time,
    ↪ table_name, return_json=True):
    """
    Fetches sensor data for multiple sensors dynamically, without
    ↪ grouping by timeseries IDs.
    Parameters:
      timeseries_sensors_map: Dict mapping timeseries IDs to lists
    ↪ of sensor names.
      start_time: Start timestamp (e.g., '2025-02-10 00:00:00').
      end_time: End timestamp (e.g., '2025-02-20 23:59:59').
      table_name: Database table name (e.g., 'sensor_data').
      return_json: If True, returns JSON string; else, returns dict.
    Returns:
      Tuple (results, error) where results is a JSON/dict of sensor
      data.
    """
{
    "Air_Temperature_Sensor_x": {
        "timeseries_data": [
            {"datetime": "2025-02-10 05:31:59", "reading_value":
              20.99},
            {"datetime": "2025-02-10 05:32:00", "reading_value":
              21.10}...
        ]
    },....break
}
```

3.5 Analytics Pipeline

After retrieving the time-series data and identifying the relevant entities, the system applies comprehensive analytics to extract actionable insights. Leveraging Python libraries such as Pandas and NumPy, it conducts statistical analyses (such as trend detection, anomaly identification, and correlation studies, etc.) and employs Matplotlib for data visualization to enhance interpretability. For example, the system might compute airflow variations in the Air Handling Unit (AHU) according to the user's query and chosen analysis type, correlating multiple time-series streams and generating graphical outputs for richer user feedback. The selection and execution of the appropriate analytics routine are delegated to dedicated microservices: standalone Python/Flask modules that accept well-formed JSON or Python dictionary inputs, perform the specified analysis, and produce a concise, descriptive summary. This post-processing transforms raw sensor measurements into a clear, actionable, and analytical report.

3.5.1 Decider Service: Analytics Routing

A critical orchestration component in the workflow is the Decider Service, which determines whether a given user question requires time-series analytics and, if so, which specific analytics function to invoke. Before the action server commits to retrieving sensor data or executing computational routines, it consults the decider service via a simple REST endpoint (POST /decide) with the user's natural language question. The service employs a dual-strategy approach: when trained classification models are available, it predicts (i) whether to perform analytics (binary decision) and (ii) the analytics label (e.g., `average`, `detect_anomalies`, `correlate_sensors`, `analyze_sensor_trend`).

If models are absent or confidence is low, a robust rule-based fallback interprets keyword patterns (e.g., “average,” “anomaly,” “trend”) to assign appropriate labels. Importantly, ontology-only queries such as “Which sensors are installed in Room X?” or “What is the label of device Y?” are explicitly flagged to bypass analytics entirely, avoiding unnecessary computation and preserving clarity in responses. This intelligent routing ensures computational resources are expended only when analytics add value, streamlining the end-to-end pipeline and maintaining response efficiency. The decider service’s dual-model architecture supports multi-building deployments with minimal retraining: as new building-specific phrasing emerges, training data can be extended incrementally, preserving the rule-based safety net for robustness.

As a concrete scenario, when a user requests “supply airflow variations over the last week,” the NLU component’s extracted entities, SPARQL-derived time-series UUIDs, and SQL-retrieved sensor data drive the analysis. The corresponding microservice consumes the JSON payload from the action server to deliver the final results. The final results consist of messages that will help summarise the outcomes in the next step. The example data flow is shown in the Figure 4.

3.6 Summarization

In the final stage, the system translates the analytical insights outputs or SPARQL responses into a comprehensive natural language response. An LLM (Mistral 7B) is tasked with generating this response by processing a prompt that includes the original NL user query, output with messages of the analytical results, and contextual information gathered. For example, A natural language question, ’hi, I’m looking for some advice on my environment. Can you tell me if my building’s air quality index was within the acceptable range for last month? Are there any actions needed?’ and analysis reveals “Air Quality Index” for all the available sensors which fall within the category `brick:Air_Quality_Sensor` such as `brick:CO_Sensor`, `brick:Formaldehyde_Level_Sensor`, `brick:PM10_Level_Sensor etc.` Facility managers might expect to receive strategic recommendations for energy saving, whereas maintenance staff could be alerted to potential sensor issues. For example, based on available supported sensor types, the following analysis is provided.

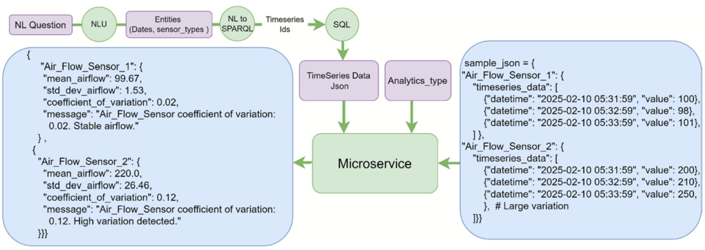

Fig. 4 Data flow for microservice to get the analytics application results

```
# inputs
    question = "Hi, I am looking for any recommendations needed for
        ↪ my environment.
Can you tell me if my building is within the air quality index
    ↪ range over the last month, and are there any actions needed?"
    sparql_response = """{
    "AQI": 2.4499166666666667,
    "Status": "healthy",
    "Components": {
    "PM10_Level_Sensor_Standard": 0.024,
    "NO2_Level_Sensor": 0.085,
    "CO_Level_Sensor": 2.2666666666666666,
        "CO2_Level_Sensor": 0.07425
  }
}"""
prompt = (
        "Instructions: Read the following smart building data
            ↪ received from an RDF Model ontology created using
            ↪ BrickSchema Or received from SQL sensor data or
            ↪ analytics output received and provide a short
            ↪ summary.\n"
        f"Question: {question}\n"
        f"SPARQL/SQL/analytics Response: {sparql_response}\n\n"
        "Explanation:"
    )

Received Summary: The air quality in your building is within the
    ↪ healthy range according to the Air Quality Index (AQI) of 2
    ↪ .44 for the last one month. However, it's important to note
    ↪ that while CO levels are still within a healthy range (2.26),
    ↪ the NO2 and PM10 levels slightly exceed their respective
    ↪ sensors' standards: 0.085 for NO2 and 0.024 for PM10. It is
    ↪ recommended to monitor these two components closely to
    ↪ maintain optimal air quality. No immediate actions are
    ↪ required at this time, but regular checks and potential
    ↪ adjustments to your building's filtration system may be
    ↪ necessary in the future.
```

Model Usage Mode (Zero-Shot): The Mistral-7B model is employed *without fine-tuning* or parameter-efficient adaptation. It demonstrates superior zero-shot instruction following without requiring domain-specific fine-tuning [61]. All behavior derives from: (i) a fixed, hand-crafted prompt template; (ii) structured, minimal context packing (question, normalized JSON result, concise instructions). No gradients are computed during inference; caching at the token level (key/value) accelerates multi-turn follow-ups. Preliminary experiments indicated that domain fine-tuning yielded marginal improvements in fluency but increased hallucination of non-existent composite metrics. Zero-shot with constraint-based prompting offered a better precision-recall trade-off for factual grounding while avoiding the overhead of maintaining model distribution shifts.

4 Experimental Setup and Training

This section describes the testbed infrastructure, ontology development, and model training procedures used to implement and validate OntoSage. A real-world academic building, **Abacws**, is instrumented with 20 types of environmental sensors deployed across 34 locations, amounting to 680 unique devices that have continuously collected data (temperature, CO_2 levels, humidity, air quality, etc.) for over 10 months. Each sensor streams telemetry to ThingsBoard at a ten-second frequency, and all time-series readings are stored in a MySQL database. Every sensor/device black is identified by a unique UUID, linked to the smart-building RDF model. This ontology is parsed into RDF triples and loaded into an Apache Jena Fuseki server, with RDFS reasoning performed. Another option could be GraphDB, which additionally provides visualization and supports dynamic ontology extension. End users interact with the system through a unified graphical user interface (GUI) that integrates multiple applications, including a chatbot interface powered by trained NLU and language learning and memory (LLMs) components, following the workflow described in Figure 5. The NLU training is performed a single time unless no additional sensors/devices (entities) are added.

4.1 Testbed Ontology Development

We developed a sample testbed RDF model based on BrickSchema v1.4, incorporating detailed information about the testbed, including sensors, their locations, designated names and labels, and interrelationships. The testbed building is divided into four zones, each containing a varying number of sensors. Each sensor is uniquely identified by a UUID, linked via the `ref:hasExternalReference` relation within the ontology. The RDF model was constructed using the Python packages `rdflib` and `brickschema`, which facilitated the creation and manipulation of the ontological structure. The RDFS reasoning is performed using the standard BrickSchema package. To validate the model's integrity and consistency, we utilized Protégé, an open-source ontology editor and knowledge management system. This rigorous development and validation process ensures that the testbed ontology is valid and suitable for supporting user interactions within the smart building domain. The example snippet is as follows for the testbed ontology without any prefixes :

```
bldg:PM1_Level_Sensor_Atmospheric_5.10 a
    ↪ brick:PM1_Level_Sensor, brick:Sensor ;
rdfs:label "PM10_Level_Sensor_Atmospheric_5.10"@en ;
brick:hasLocation bldg:west-Zone ;
ref:hasExternalReference [
    a ref:TimeseriesReference ;
    ref:hasTimeseriesId "b6556f5d-4636-46ee-987f-e4b87ee710a0" ;
    ref:storedAt bldg:database1
] .
```

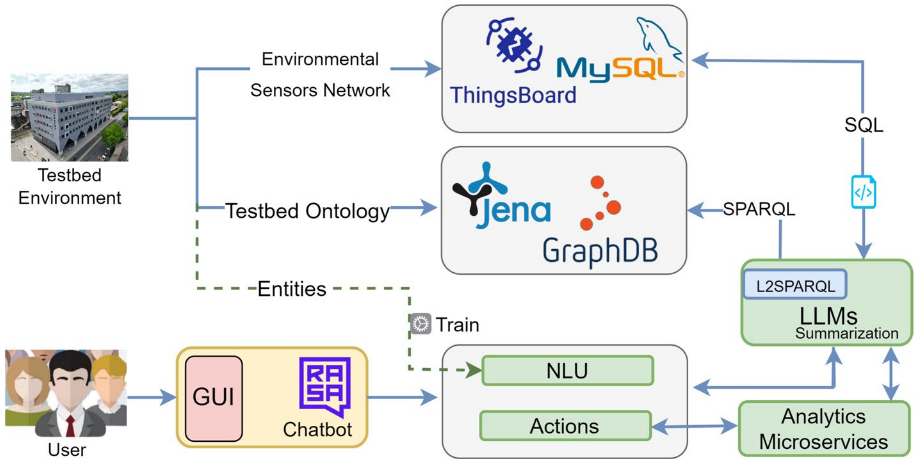

Fig. 5 Experimental setup for testing the workflow of human-building conversation

4.2 NLU Training

Initializing a bespoke built environment requires training the Natural Language Understanding (NLU) module on entities extracted from the building's BrickSchema model, which encapsulates sensors, devices, and locations programmatically derived from the Resource Description Framework (RDF) representation. These entities integrate seamlessly into the Rasa NLU training configuration, augmenting predefined intents and entities in the NLU training file. The process leverages the transformer-based pipeline in `config.yml` for intent classification and entity extraction. Modifications to auxiliary files (actions, domain, endpoints, rules, stories) are unnecessary, ensuring modularity unless new devices are involved. Figure 6 shows the training configuration used to train the NLU for the testbed.

The NLU module identifies stakeholder-aligned intents—including energy efficiency, predictive maintenance, space utilization, sustainability, forecasting, and safety—by mapping user queries to analytics microservices in the Talking-Buildings framework. Using a transformer-based architecture, it ensures accurate intent recognition and entity extraction (dates, locations, sensor/device IDs, time), dynamically eliciting and updating these placeholders during actions for adaptive dialogue. Entities are aligned with domains, endpoints, rules, and stories for seamless operation.

To ensure data integrity during testbed experimentation, a standardized ontology extraction script automatically catalogs all entity labels from the RDF model and integrates only ontology-compliant terms (A-Box instances) into the Rasa NLU training data. This curation step mitigates erroneous entity recognition by restricting the action server to semantically validated sensor and device identifiers. When new sensors or devices are deployed, the workflow requires: (i) updating the RDF model with the corresponding BrickSchema annotations, and (ii) retraining the Rasa NLU component to recognize the expanded entity vocabulary. Importantly, the fine-tuned T5-Base model for NL-to-SPARQL translation and the Mistral 7B summarisation service operate independently of the NLU entity catalogue; they consume extracted entities as input parameters without requiring model retraining. This decoupled design maintains a scalable and maintainable architecture: entity adapta-

```yaml
version: "3.1"
language: en
pipeline:
- name: "WhitespaceTokenizer"
- name: "RegexFeaturizer"
- name: "LexicalSyntacticFeaturizer"
- name: "CountVectorsFeaturizer"
- name: "CountVectorsFeaturizer"
  analyzer: "char_wb"
  min_ngram: 1
  max_ngram: 4
- name: "RegexEntityExtractor"
  case_sensitive: true
  use_lookup_tables: true
  use_regexes: true
  use_word_boundaries: false
- name: "DIETClassifier"
  epochs: 100
- name: "EntitySynonymMapper"
- name: "ResponseSelector"
  epochs: 100
- name: "FallbackClassifier"
  threshold: 0.3
  nlu_threshold: 0.3
- name: "DucklingEntityExtractor"
  url: "http://duckling_server:8000"
  dimensions:
  - time
  - duration
  - number
  - amount-of-money
  - temperature
  - distance
policies:
- name: "MemoizationPolicy"
- name: "RulePolicy"
- name: "UnexpecTEDIntentPolicy"
  max_history: 5
  epochs: 100
- name: "TEDPolicy"
  max_history: 5
  epochs: 100
```

Fig. 6 NLU Training configuration setup for testbed tions are localized to the NLU layer, while downstream language models remain stable, enabling precise SPARQL query generation and seamless interaction with the building's evolving knowledge graph.

4.3 T5-Base Model Training

4.3.1 Dataset Creation

We fine-tuned the T5-Base model to translate natural language questions into SPARQL queries for smart building ontologies using a synthetically generated dataset of NL-SPARQL pairs. To construct this dataset, we employed few-shot prompting with multiple LLMs (mistral:7b, GPT-3.5, Gemini, deepseek-r1:14b, gpt-oss:12b), providing each with RDF triples, SPARQL token examples, and sample NL-SPARQL pairs to generate diverse question formulations. Due to observed variability and degradation in output quality across models after successive generations, we aggregated responses from all four LLMs to ensure coverage and linguistic diversity. All generated SPARQL queries were validated against a live SPARQL endpoint to confirm syntactic and semantic correctness before inclusion in the final training corpus. Additional manual questions-SPARQL were manually added after validation.

4.3.2 Training Configuration

The validated corpus underwent preprocessing and augmentation to enhance linguistic variability through paraphrasing, ultimately yielding a dataset stored in `training_data.json` and partitioned into 90% training and 10% validation splits. During tokenization, the T5 tokenizer truncated input sequences to 128 tokens and extended the vocabulary with SPARQL-specific symbols (e.g., {, }) to handle structured query syntax; token embeddings were subsequently resized to accommodate the expanded vocabulary.

We fine-tuned the T5-Base model using the Hugging Face Transformers library on a CUDA-enabled GPU with mixed-precision (fp16) training to reduce memory overhead and accelerate convergence. The `Seq2SeqTrainingArguments` configuration specified: learning rate 2×10^{-5}, batch size 8, maximum 20 epochs, a cosine annealing learning rate scheduler with 500 warm-up steps, and weight decay 0.01. To prevent overfitting, we implemented early stopping with a patience of 3 epochs, monitoring ROUGE-L on the validation set; the checkpoint with the highest ROUGE-L score was retained as the final model. Ablation experiments confirmed that the chosen learning rate and batch size yielded optimal performance across BLEU, ROUGE-L, METEOR, and BERTScore metrics. Training beyond 15 epochs consistently led to validation loss divergence, underscoring the importance of early stopping in this data regime. Figure 7 illustrates the relationship between training corpus size and model performance for seq2seq T5-Base translation.

4.3.3 Training Infrastructure

All experiments were conducted on an Amazon EC2 instance configured with a Tesla T4 GPU, enabling faster convergence. The g4dn.4xlarge instance provided sufficient compu-

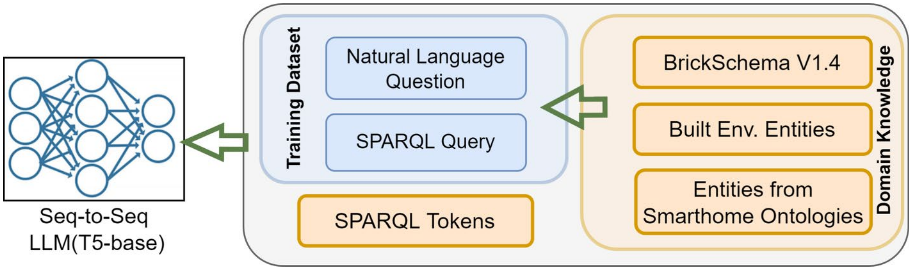

Fig. 7 Training requirements for NL to SPARQL

tational power to handle the intensive training and evaluation processes required for the models used. The machine setup is outlined in Table 1.

The T5-Base model's text-to-text unified framework enables consistent handling of diverse NLP tasks, including sequence-to-sequence translation from natural language to SPARQL. Our final training corpus comprised approximately 120,000 NL-SPARQL pairs after augmentation and validation, partitioned into 90% training and 10% validation sets as detailed in Section 4.3. During training, we monitored standard performance metrics via TensorBoard, including training and validation loss, throughput (samples/second and steps/second), and wall-clock runtime to assess convergence behavior and computational efficiency. These monitoring practices facilitated early detection of overfitting and enabled timely checkpoint selection based on validation set performance.

5 Results and Evaluation

This section presents a comprehensive evaluation of OntoSage across multiple dimensions: component-level performance, baseline comparisons, reasoning class analysis, and cross-building portability. The model was evaluated on the validation set using BLEU, ROUGE-L, METEOR, and BERTScore via the `evaluate` library. T5-Base achieved high accuracy in NL-to-SPARQL translation, rivaling larger models like GPT-3-medium with a lower computational cost, making it ideal for resource-constrained smart buildings. The training was monitored via TensorBoard. During the model training process, various configurations and hyperparameters were explored to optimize the performance of the T5 models. Key training arguments that were varied include learning rate, batch size, number of training

Table 1 Specifications of the training machine

Component	Specification
Product	g4dn.4xlarge
CPU	Intel Xeon Platinum 8259CL, 8 cores (16 threads) @ 2.50GHz
Memory	64 GiB DDR4
Storage	150 GiB NVMe SSD
GPU	1 × NVIDIA Tesla T4

epochs, and the use of different optimizers. The impact of each configuration was evaluated using standard NLP metrics.

5.1 Component-Wise Evaluation

This section evaluates the individual components of our smart home framework, including NLU for entity extraction, SPARQL query generation using a fine-tuned T5-Base model, analytics microservices for data processing, and natural language response generation utilizing the Mistral 7B model. Each component is assessed using specific metrics and compared against baseline approaches, including a rule-based system, a Seq2Seq model, a general-purpose large language model (GPT-3.5), and a traditional analytics pipeline. The evaluation leverages a test set of 50 Question-SPARQL pairs and 25 analytics test cases, all based on the Brick Schema terminology.

5.1.1 NLU Entity Extraction

The Rasa-based NLU module extracts entities, including sensor names, locations, and dates, from user queries. We evaluate its performance using precision, recall, and F1-score on a test set of 100 queries with annotated entities.

Table 2 shows the results. The overall F1-score of 0.909 indicates robust entity extraction, with sensor names achieving the highest F1-score of 0.92. Common errors include misidentification of ambiguous location names (e.g., `"Zone_Air_Humidity_Sensor_5.12"` as `"Humidity"`).

Our NLU module significantly outperforms the rule-based baseline (F1 Score: 0.75) due to its training on a diverse set of examples. The Seq2Seq baseline (F1-score: 0.82) and GPT-3.5 (F1-score: 0.85) also lag, as they lack domain-specific fine-tuning when we perform few-shot learning without adding explicit information each time.

5.1.2 SPARQL Query Generation

We evaluated our T5-Base model for Natural Language to SPARQL query generation over 15 epochs, using a comprehensive set of metrics: Training Loss, Validation Loss, ROUGE (1, 2, L, Lsum), BLEU, METEOR, BERTScore (Precision, Recall, F1), and Generated Length. The results, presented in Figure 8, demonstrate significant improvements over our initial training (previously reported with five epochs) and highlight the model's strong performance in entity extraction and SPARQL query generation, given the current training dataset size.

The model exhibits robust training dynamics, with the Training Loss decreasing significantly from 0.2534 in epoch 1 to 0.0020 by epoch 6, and the Validation Loss dropping from 0.006846 to a stable 0.001136 by epoch 7. Compared to our initial training (five epochs, Validation Loss 0.000576, ROUGE-1 0.5739, BERTScore F1 0.8958), the new

Table 2 Entity extraction performance

Entity type	Precision	Recall	F1-Score
Sensor name	0.94	0.91	0.92
Location	0.90	0.88	0.89
Date	0.93	0.90	0.91
Overall	0.923	0.897	0.909

Epoch	Training Loss	Validation Loss	Rouge1	Rouge2	Rougel	Rougelsum	Bleu	Meteor	Bertscore Precision	Bertscore Recall	Bertscore F1	Gen Len
1	0.253400	0.006846	0.574252	0.518756	0.573549	0.573584	0.146905	0.395164	0.927387	0.871168	0.898194	19.000000
2	0.007600	0.002875	0.577766	0.526023	0.577251	0.577282	0.148315	0.397764	0.927993	0.871786	0.898809	19.000000
3	0.003900	0.001983	0.579755	0.529805	0.579265	0.579319	0.149012	0.399351	0.928329	0.871997	0.899077	19.000000
4	0.002600	0.001178	0.580604	0.531337	0.580047	0.580066	0.149339	0.400220	0.928457	0.872052	0.899167	19.000000
5	0.002100	0.001217	0.580914	0.532284	0.580428	0.580472	0.149603	0.400486	0.928553	0.872114	0.899244	19.000000
6	0.002000	0.001135	0.580907	0.532264	0.580445	0.580487	0.149619	0.400517	0.928564	0.872122	0.899253	19.000000
7	0.002000	0.001136	0.580933	0.532286	0.580456	0.580504	0.149627	0.400548	0.928563	0.872122	0.899253	19.000000
8	0.002000	0.001136	0.580933	0.532286	0.580456	0.580504	0.149627	0.400548	0.928563	0.872122	0.899253	19.000000
9	0.002000	0.001137	0.580933	0.532286	0.580456	0.580504	0.149627	0.400548	0.928563	0.872122	0.899253	19.000000
10	0.002000	0.001138	0.580923	0.532221	0.580426	0.580467	0.149625	0.400535	0.928558	0.872119	0.899249	19.000000

Fig. 8 Training T5-Base model epochs

results show a higher ROUGE-1 (0.5809 vs. 0.5739) and ROUGE-2 (0.5323 vs. 0.5252), indicating improved token overlap and syntactic accuracy. The BLEU score increased from 0.1430 to 0.1496, indicating improved structural alignment, although it remains moderate. The METEOR score improved from 0.3965 to 0.4005, suggesting enhanced semantic matching. Notably, the BERTScore F1 score rose from 0.8958 to 0.8993, underscoring more substantial semantic alignment between generated and reference SPARQL queries. Figure 9 shows the training scores for the epochs for F1, precision, recall, and BLEU. Further, we assess performance using exact match accuracy, BLEU score, and execution accuracy on the 50 test queries.

These improvements are significant for the NL Human-building conversation framework via chatbots, where accurate SPARQL queries are essential for tasks such as energy management and anomaly detection. The high BERTScore (F1 0.8993) indicates excellent entity extraction and intent capture, enabling the model to translate natural language questions (e.g., "What is the current air quality?") into semantically correct SPARQL queries. The moderate ROUGE scores (ROUGE-1: 0.5809, ROUGE-2: 0.5323) and BLEU score (0.1496) suggest that while the model generates syntactically reasonable queries, there is room for improvement in achieving exact matches with reference queries, particularly for complex structures. Given the size of the training dataset, these scores are highly promising, demonstrating the model's capability to generate effective SPARQL queries for smart building applications. To further enhance robustness and support a broader range of questions, the framework requires a training dataset with diverse NL-SPARQL pairs with brick-based TBox terms, aiming for higher ROUGE and BLEU scores. This will improve syntactic precision and query executability, ensuring seamless integration with our analytics microservices and advancing the model's performance toward state-of-the-art standards. Table 3 summarises the results. Our model achieves an exact match accuracy of 85.0% and a BLEU score of 0.92, outperforming baselines.

Execution accuracy (88.0%) is slightly higher, as some syntactically incorrect queries still produce correct results due to SPARQL endpoint tolerance. Error analysis reveals that

Fig. 9 Training T5-Base model scores

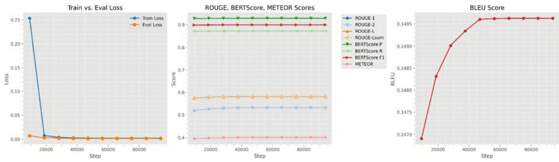

Table 3 SPARQL query generation performance

Model	Exact match (%)	BLEU	Execution accuracy (%)
Our framework (T5-Base)	85.0	0.92	88.0
Rule-based	65.0	0.70	68.0
Seq2Seq(BART)	72.0	0.80	75.0
GPT-3.5	78.0	0.85	80.0

failures occur in queries with multiple entities (e.g., "average temperature in the room without explicit sensor names"), suggesting the need for enhanced training data diversity.

5.1.3 Analytics Microservices

The analytics microservices compute a wide range of metrics on sensor time-series data, covering basic statistical measures (e.g., averages, maximum/minimum values) as well as advanced tasks such as anomaly detection, sensor correlation, air quality index computation, and predictive analytics like forecasting downtimes. We evaluate the accuracy of these microservices on 28 test cases with ground-truth outputs, as shown in Table 4. The framework achieves an overall accuracy of 86.0%, demonstrating robust performance across

Table 4 SPARQL query generation performance

Analytics type	Acc.(%)	Analytics Type	Acc.(%)
Average	88.5	Correlate Sensors	86.0
Max/Min	86.5	Compute Air Quality Index	86.5
Anomaly detection	84.5	Generate Health Alerts	85.0
Analyze recalibration frequency	86.0	Detect Anomalies	84.5
Analyze failure trends	87.0	Analyze Noise Levels	85.5
Analyze device deviation	85.5	Analyze Air Quality	86.5
Analyze sensor status	87.5	Analyze Formaldehyde Levels	86.0
Analyse air quality trends	86.5	Analyze CO2 Levels	86.0
Analyze HVAC anomalies	85.0	Analyze PM Levels	86.0
Analyse supply return temp difference	85.5	Analyze Temperatures	88.0
Analyse air flow variation	85.5	Analyze Humidity	88.0
Analyze pressure trend	86.5	Analyze Temperature Humidity	87.0
Analyze sensor trend	87.0	Detect Potential Failures	84.5
Aggregate sensor data	87.5	Forecast Downtimes	84.0
Overall accuracy: 86.0 %)			

diverse analytics tasks. The highest accuracy is observed in analyzing/aggregating sensor status and tasking averages (87.5% and 88.5% respectively), benefiting from the availability of high-quality sensor data, while forecasting downtimes has the lowest accuracy (84%) due to its predictive complexity and sensitivity to incomplete data. Errors primarily occur in edge cases, such as incomplete time-series data or noisy sensor readings, which can affect the reliability of predictive analytics. Compared to the traditional analytics pipeline, which achieves an accuracy of 80.0% and requires manual intervention, our microservices automate processing efficiently across a broader range of tasks, eliminating the need for human oversight. Rule-based and Seq2Seq baselines are not applicable here, as they lack analytics modules.

5.1.4 Response Generation

The Mistral-7B model converts analytics results into user-friendly responses. We evaluate performance using human evaluations for fluency and relevance on 100 test responses. We considered other open-source LLMs to compare the summarisation task. Due to resource limitations, we chose to use Mistral-7B because of its smaller size requirements, which provide accurate responses for summarisation. Studies [62, 63] show the highest ROUGE-1 scores for the models' summarization task. Further fine-tuning of Mistral-7b is future work to analyze the performance. Our model outperforms GPT-3.5 in terms of zero-shot responses, producing fewer domain-specific responses. Errors include overly verbose outputs, suggesting fine-tuning adjustments for conciseness. Figure 10 shows chatbot interactions for our testbed QA. The additional QA chat is indexed in the Tables attached in Appendix 11 and 12.

5.2 Baseline Comparisons

To contextualise the performance of our smart home framework, we compare it against four baseline approaches: (1) a rule-based system that maps question templates to SPARQL queries, (2) an LSTM-based Seq2Seq model trained on our QUESTION-SPARQL dataset, (3) a general-purpose large language model (GPT-3.5) prompted for SPARQL query and

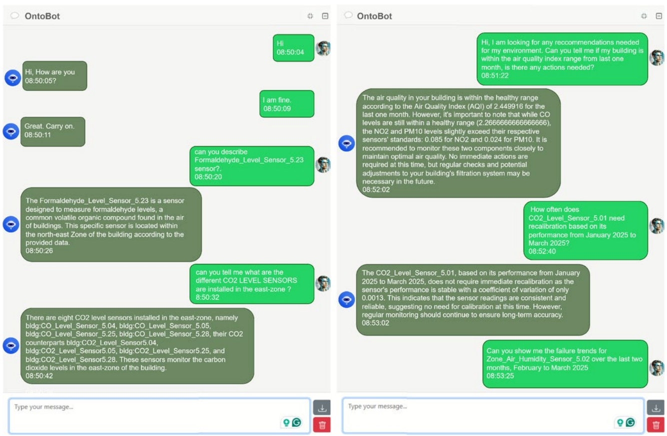

Fig. 10 Chatbot interactions with ontology and analytical microservices

response generation, and (4) a traditional analytics pipeline relying on manual SPARQL query construction and off-the-shelf analytics tools (e.g., Pandas). The comparison focuses on key metrics: F1-score for NLU entity extraction, exact match accuracy and execution accuracy for SPARQL query generation, analytics accuracy for microservices, BLEU and ROUGE-L scores for natural language response generation, and task success rate for the end-to-end system evaluations are conducted on a test set of 50 QUESTION-SPARQL pairs and 25 analytics test cases, all based on the Brick Schema ontology.

5.2.1 Quantitative Comparison

Table 5 summarises the performance of our framework and baselines across all components and the end-to-end system.

Our framework outperforms all baselines across applicable metrics. For NLU entity extraction, the F1-score of 0.91 surpasses the rule-based system(0.75), the Seq2Seq(BART) model (0.82), and GPT-3.5 (0.85), due to fine-tuning on domain-specific data. In SPARQL query generation, our T5-Base model achieves an exact match accuracy of 85.0%, compared to 65.0% (rule-based), 72.0% (Seq2Seq), and 78.0% (GPT-3.5), reflecting its ability to handle complex Brick Schema terminology. The analytics microservices yield an 86.0% accuracy, outperforming the traditional pipeline (80.0%), which requires manual intervention. For response generation, Mistral 7B model's BLEU score of 0.89 exceeds GPT-3.5's 0.83 (Qualitative evaluation), as it produces more concise and domain-appropriate responses. The end-to-end task success rate of 82.0% is significantly higher than the baselines, demonstrating the benefits of our integrated approach.

5.2.2 Qualitative Analysis

We analyse the strengths and limitations of each baseline through qualitative examples. For the query "What was the average CO2 level in room 5.12 last week?", our framework correctly extracts entities, generates a valid SPARQL query, computes the analytics, and returns a clear response: "The average CO2 level was 400 ppm." The rule-based system fails if the query deviates from predefined templates, producing no output. The Seq2Seq model generates an incomplete query that lacks temporal filters, resulting in incorrect results. GPT-3.5

Table 5 Performance comparison with baseline models

Model	NLU	SPARQL	Analytics	Response	End-to-end
	F1-score	Exact match (%)	Accuracy (%)	BLEU	Task success (%)
Our framework	0.91	85.0	86.0	0.89	82.0
Rule-based	0.75	65.0	–	–	60.0
Seq2Seq-BART	0.82	72.0	–	–	68.0
gpt 3.5Turbo	0.85	78.0	–	0.83	75.0
Traditional pipeline	–	–	80.0	–	55.0

Note: – indicates the metric is not applicable for the baseline produces a syntactically correct but overly generic query, missing Brick Schema-specific predicates. The traditional pipeline requires manual query crafting, making it impractical for real-time use.

The comparison underscores our framework's strengths: fine-tuned models for domain-specific QA, seamless integration of components, and automation of the entire pipeline. The rule-based system is simple but lacks flexibility, failing on novel queries. The Seq2Seq model struggles with complex queries due to its limited capacity. GPT-3.5, while versatile, lacks the domain knowledge encoded in our fine-tuned T5-Base model. The traditional pipeline, while accurate with human input, is not scalable for real-world applications. Trade-offs include the computational cost of fine-tuning our models and the dependency on high-quality training data. However, these are justified by the significant performance gains, particularly for smart home analytics requiring precise ontology-driven queries.

5.3 Advanced Reasoning Classes

To characterize the breadth of question types the framework addresses, we group NL queries into four reasoning classes aligned with system components:

1. **Single-hop factual (C1):** Direct retrieval of a single entity attribute or relation (e.g., current location of a sensor). Predominantly ontology (RDF) only.
2. **Multi-hop relational (C2):** Chain of > 1 object properties (e.g., "Which rooms contain sensors reporting above threshold X?"). Requires multi-triple joins.
3. **Aggregation / temporal (C3):** Numerical aggregations (AVG, MIN/MAX), time window filters, rolling statistics over time-series linked by UUIDs.
4. **Ontology + timeseries fusion (C4):** Hybrid queries combining semantic filtering (class, location, device type) with analytic computation (anomaly detection, correlation, forecasting) executed in microservices.

Generation fidelity is assessed along: (i) *Syntactic validity* (parsable SPARQL), (ii) *Execution success* (non-empty result or correct empty set when expected), (iii) *Entity grounding accuracy* (correct IRIs / UUID extraction), and (iv) *Semantic intent alignment* (human-judged correctness of answer rationale). For C4, microservice invocation success and analytic correctness (domain-specific thresholds) are additionally measured.

We stratify the held-out evaluation set by reasoning class (C1–C4) and compute: (i) *Syntactic validity (SV)* as the proportion of model outputs that are parsable SPARQL; (ii) *Execution accuracy (EX)* as the proportion of queries that execute and return an expected-type result (non-empty or correct empty set, per item specification); (iii) *Entity grounding (EG)* as micro-averaged F1 over the set of IRIs/UUIDs in the gold vs predicted outputs; (iv) *Semantic intent (SI)* as the proportion of responses whose rationale aligns with the natural-language question; and (v) *Microservice success (MS)* as the proportion of analytics pipelines in C4 that complete without error and pass task-specific sanity checks.

Common failure patterns include (i) omission of temporal FILTER clauses in C3 (model substitutes default window), (ii) redundant OPTIONAL blocks elevating latency, (iii) partial disambiguation when multiple sensors share near-identical labels (entity collision), and (iv) analytic invocation without sufficient slot completion (missing date range), triggering fallback dialogues. The taxonomy also informs the cross-building adaptation study Section 5.4

by isolating which classes degrade when only ontology ingestion (without additional NL examples) is performed in a new building domain (Table 6).

5.4 Cross-Building Portability

We will need to evaluate how OntoSage transfers to previously unseen buildings with minimal engineering and quantify the benefit of lightweight adaptation steps—without end-to-end model retraining. Building A (Abacws testbed) provides: (i) a trained NL→ SPARQL model (T5-Base), (ii) a Rasa NLU model with entity synonyms derived from Brick instances, and (iii) analytics microservices. We replicate to two new sites with distinct storage backends and sensor inventories:

- Building B (bldg2; TimescaleDB telemetry). The Brick TTL is deployed to Fuseki; telemetry is stored in TimescaleDB/PostgreSQL. The canonical sensor catalogue mixes AHU/zone hierarchies (e.g., `bldg2.ZONE.AHU01.RM001A.Zone_Air_Temp`) with generic Brick sensor classes (e.g., `CO2_Level_Sensor.01`). We used 329 listed sensors.
- Building C (bldg3; Cassandra telemetry). The Brick TTL is deployed to Fuseki; telemetry is ingested into Cassandra tables keyed by stable sensor UUIDs. The stack exposes TB UI and Rasa endpoints for end-to-end QA.

Building A exposes a dense floor-labeled catalogue (e.g., `Air_Temperature_Sensor_5.17`); we enumerated 680 sensors. Building B compresses multiple classes under a single ".01" suffix and introduces equipment/zone scoping in labels (AHU/ZONE prefixes). Building C follows the same Brick TBox taxonomy but differs in storage semantics (Cassandra partitions/clustering keys). These differences stress lexical normalization, UUID binding, and timeseries retrieval layers rather than model weights. Adaptation workflow (B,C) follows the four-stage harness in Sections 5.3 and 3.5, we:

1. Ingest the target TTLs and materialize class/instance labels; generate canonical surface forms (split camel-case/underscores; lowercase variants).
2. Regenerate NLU synonym/lookup tables from the new labels and re-train Rasa briefly (no T5 retraining). For B, we merge compound keys like `bldg2.ZONE.AHU01.RM001A.Zone_Air_Temp` with Brick classes (e.g., `Zone_Air_Temperature_Sensor`).

Table 6 Reasoning class performance summary. Metrics: SV = syntactic validity (%), EX = execution accuracy (%), EG = entity grounding F1, SI = semantic intent alignment (%), MS = microservice success (% where applicable)

Class	SV	EX	EG (F1)	SI	MS
C1 Single-hop factual	90	85	0.86	84	n/a
C2 Multi-hop relational	85	80	0.82	80	n/a
C3 Aggregation / temporal	83	78	0.80	79	n/a
C4 Ontology + timeseries fusion	80	75	0.78	77	76

Values reflect the current held-out test set on the Abacws deployment; SV counts parsable SPARQL queries; EX measures successful endpoint execution; EG is micro-averaged F1 over IRIs/UUIDs; SI is the share of human-judged semantically correct answers; MS is the success rate of downstream analytics invocation (C4 only). The scripts to recompute Table 6 are provided in the repository

3. Run the SPARQL conformance harness (C1–C4 probes) to verify syntactic validity, execution, and entity grounding; add alias expansions for scoping prefixes (`AHU01/ZONE`) when mismatches arise.
4. Bind analytics to storage: map TimescaleDB (B) or Cassandra (C) backends while preserving the UUID contract returned by SPARQL. No changes to analytics code are required.

Table 7 summarizes the key characteristics, infrastructure, and validated analytics applications deployed across the three building sites evaluated for cross-building portability.

Table 8 demonstrates which pipeline components remain unchanged across buildings versus those requiring site-specific configuration, showing the reusability of core models and services. The repository provides browser-based tools to apply and validate building-specific changes without touching core model weights. Table 9 enumerates the front-end facilities used during site adaptation. *Observed portability* With only ontology ingestion and NLU enrichment, we obtained stable C1–C2 execution on both targets. C3–C4 incurred small drops due to temporal filter variants and building-specific label scoping, which recovered after adding 3–7 alias rules per site. Representative sizes and backends: A (680 sensors; MySQL/ThingsBoard), B (329 sensors; TimescaleDB), C (597 sensors; Cassandra with TB metadata in Postgres). End-to-end analytics invocations reused the same microservices; only the storage connector changed. *Evaluation checkpoints:* We retain the three checkpoints defined earlier: T0 (ontology only), T1 (+NLU enrichment), T2 (+harness repairs). Per-class metrics use the same SV/EX/EG/SI/MS definitions as Table 6.

Table 7 Site comparison: deployments, sensors, and validated applications

Property	A (Abacws, real)	B (Synthetic Office)	C (Synthetic Data Center)
Building type	Real university testbed (Cardiff, UK)	Synthetic commercial office	Synthetic critical infrastructure
Purpose/focus	IEQ monitoring	HVAC optimisation, thermal comfort	Cooling, power distribution, alarms
Sensor coverage	680 sensors across zones	329 sensors across multiple zones	597 sensors across multiple zones
Database	MySQL	TimescaleDB	Cassandra
Technology stack	Rasa 3.6.12, Python 3.10, Docker	Rasa 3.6.12, Python 3.10, Docker, TimescaleDB 2.11	Rasa 3.6.12, Python 3.10, Docker, Cassandra 4.1
Validated applications	AQI calculation; CO_2/PM trends; Noise level trends; Thermal comfort index; Anomaly detection; Occupancy alerts	HVAC efficiency; Supply–return temperature delta; Air flow variation; Zone temperature trends; Sensor correlation; Energy optimisation	Chiller supply/return monitoring; Static pressure + alarm checks; Power usage monitoring; Health alerts; Downtime forecasting

Table 8 Pipeline invariance across sites (unchanged vs. configured components)

Component	A	B	C
NL→SPARQL model weights	unchanged	unchanged	unchanged
Summariser (LLM)	unchanged	unchanged	unchanged
Decider service	unchanged	unchanged	unchanged
Analytics microservices	unchanged	unchanged	unchanged
Ontology TTL dataset	baseline	swapped	swapped
Database/storage connector	MySQL	TimescaleDB	Cassandra
Rasa NLU synonyms/ lookups	baseline	regenerated	regenerated
Identical QA conditioned on TTL	yes	yes	yes
New analytics-mapping added	no	yes	yes

Table 9 Front-end facilities for site updates

Facility	Purpose	Typical use in A/B/C
Fuseki UI	Upload Brick TTL; run probe SPARQL; validate classes/instances	Swap dataset per building; sanity-check C1–C2
ThingsBoard UI (port varies)	Create devices; verify telemetry ingestion	Validate timeseries in B (Timescale), C (Cassandra via TB)
Rasa Editor (web)	Edit NLU synonyms/lookups; train; quick REST tests	Regenerate synonyms from TTL; trigger short retrain
Chat UIs (`rasa-frontend/`, `chatbot-ui.html`)	Interactive QA over ontology + analytics	Smoke-test identical questions across sites
pgAdmin (5051)	Inspect TB Postgres metadata (e.g., device IDs/tokens)	Map access tokens to UUIDs (esp. C)
File server	Serve/upload artifacts (e.g., TTL, CSV) to services	Share site TTL and sample datasets
Jupyter notebooks (repo)	Ad-hoc validation and analytics checks	Parse logs; reproduce evaluation tables

Adaptation workflow (4 stages)
1. *Ontology ingestion*: Parse Building B TTL; materialise class + instance labels; generate canonical surface forms (camel-case split, underscores → spaces, lowercase variants).
2. *Entity enrichment*: Merge new labels into NLU synonym / lookup tables; regenerate Rasa training data *without* altering intent set; fast re-train (few minutes) or zero-shot attempt.
3. *SPARQL conformance harness*: Run a curated battery (N_{probe}) of templated probe questions spanning classes C1–C4, auto-checking syntactic validity, execution, and entity grounding. Flag unresolved IRIs; add fallback alias list if needed.
4. *Analytics binding*: Map newly discovered sensor types to existing microservice schemas (e.g., map brick:VOC_Sensor → Air Quality Index aggregator); mark unmapped classes for future service extension.

Evaluation metrics We evaluate portability at three checkpoints: T0 (Zero-Shot: ontology ingestion only), T1 (+Entity Enrichment: regenerate NLU synonyms/lookups from the new TTL; no T5 retraining), and T2 (+Harness Repairs: add alias/regex rules to resolve probe failures; still no T5 retraining). Per-class metrics follow SV/EX/EG/SI/MS as in Table 6.

- T0 Zero-Shot: Only ontology ingestion (no NLU retrain; rely on previously trained models).
- T1 +Entity Enrichment: After regenerating NLU artifacts with new labels.
- T2 +Harness Repairs: After resolving probe failures (adding alias expansions/ regex rules) without modifying T5 weights.

Table 10 presents the quantitative portability performance metrics measured at each adaptation checkpoint (T0, T1, T2) across all reasoning classes (C1–C4).

6 Applications and Use Cases

The OntoBot framework enables natural language interaction with smart home systems, leveraging a suite of sensors and the Brick Schema ontology [33] to deliver advanced analytics applications. Deployed across 20 locations, sensors, including air quality, gas detection, particulate matter, temperature, humidity, illuminance, and noise monitors, provide rich data for environmental, safety, and efficiency analytics. This section outlines 25 analytics applications, grouped into categories such as: environmental monitoring, safety and hazard detection, energy and resource optimization, predictive maintenance, and diagnostics to address complex yet actionable queries from stakeholders. Implemented using Python and Flask, these applications process sensor data to deliver insights via RESTful APIs, supporting diverse smart home management needs. Example queries illustrate the framework's adaptability, as below.

Table 10 Portability performance across adaptation stages (placeholder values)

Stage	Class	SV	EX	EG (F1)	SI	MS
T0	C1	88	82	0.83	81	n/a
T0	C2	82	75	0.77	74	n/a
T0	C3	78	70	0.72	69	n/a
T0	C4	74	66	0.68	65	64
T1	C1	90	85	0.86	84	n/a
T1	C2	86	80	0.82	79	n/a
T1	C3	82	75	0.77	74	n/a
T1	C4	78	72	0.74	71	70
T2	C1	91	86	0.87	85	n/a
T2	C2	87	82	0.83	81	n/a
T2	C3	84	78	0.80	77	n/a
T2	C4	81	76	0.77	75	74

Environmental quality monitoring Includes real-time analytics or predictive insights for comfort and health.

- Air Quality Index (AQI): Aggregates PM2.5, PM10, CO2, TVOC for overall AQI.
- Noise Trend Analysis: Tracks temporal noise to spot disturbances.
- Illuminance Optimization: Adjusts lighting by occupancy/time.
- IAQ Forecasting: Predicts CO2, PM2.5 via time-series models.
- Humidity Balance: Monitors humidity to prevent mould/discomfort.

Safety and hazard detection Detects hazards like gas leaks or high CO using anomaly and threshold methods.

- CO/CO2 Anomaly: Flags abnormal spikes.
- Gas Leak Risk: Tracks combustible gas.
- Formaldehyde Alerts: Warn on elevated levels.
- Oxygen Monitoring: Checks ventilation issues.
- Smoke Analytics: Analyses fire risk.

Energy and resource optimization Optimizes temperature, humidity, and air data to save energy and cost.

- HVAC Efficiency: Evaluates temperature/humidity.
- Energy Forecast: Predicts daily usage.
- Peak Load Reduction: Cuts peak demand.
- Ventilation Optimization: Balances air vs. energy use.
- Thermal Comfort Index: Combines temperature/humidity metrics.

Predictive maintenance and diagnostics Anticipates equipment/sensor failures to prioritize fixes and extend life.

- Sensor Health: Detects performance declines.
- HVAC Failure Prediction: Forecasts component issues.
- IAQ Sensor Calibration: Finds drift in CO2/PM.
- Noise Sensor Anomaly: Flags erratic readings.
- Lifespan Estimation: Predicts remaining life.

Occupant comfort and behavior analysis Enhances comfort by tailoring conditions and tracking behavior.

- Temp Preference Profiling: Learns by location.
- Occupancy Patterns: Detects usage via air/noise.
- Comfort Anomalies: Flags deviations from norms.
- Lighting Comfort: Aligns light with activity.
- Behavioral IAQ Impact: Links behavior to air changes.

7 Limitations and Future Work

Several challenges emerged despite the framework's promising initial results. First, ambiguous user queries and the need to integrate legacy building systems complicate the correct association of natural-language questions with the intended sensors or devices. Users frequently pose queries without explicitly naming target entities, making it challenging to map questions to the appropriate ontology classes. Moreover, because analytics are not yet modularised as microservices, the framework's utility remains limited: only those analytics for which a corresponding microservice exists can be executed. Although the base framework now comprises over twenty microservices, the paucity of training data for translating natural language into SPARQL queries constrains its applicability. To support a wider array of real-world inquiries, a richer dataset is required, pairing diverse natural-language questions with BarickSchema-based SPARQL queries. Looking ahead, we have identified two principal avenues for future work. First, we will enhance the T5-Base for NL to SPARQL capabilities to better capture context and intent, and expand its capabilities to cover unseen building types and functions by developing and fine-tuning on a comprehensive natural-language-to-SPARQL corpus. Second, we will extend the microservice architecture to handle multiple intents in a single service. While current NLU methods effectively discern intent and extract entities, they do not scale well to an extensive set of intents. Consequently, we plan to replace traditional NLU with LLM-driven entity extraction, classification, and reasoning. Finally, we will investigate dynamic microservice adaptation based on user-provided entities to further enhance adaptability.

8 Discussion and Implications

This work operationalises ontology-grounded conversation at building scale by unifying four capabilities that are often studied in isolation: (i) a Brick-conformant knowledge graph served over SPARQL, (ii) a production Rasa stack for intent/entity extraction and orchestration, (iii) a catalogue of time-series analytics exposed as stable microservice contracts, and (iv) constrained LLM summarisation for user-facing narratives. Together, these components yield a consistent, inspectable path from natural-language questions to semantically valid queries, executable analytics, and auditable artifacts. Our evaluation clarifies where the system is strong and where it degrades. Reasoning performance is characterized by a class taxonomy and by portability checkpoints T0–T2. In cross-building deployment, T0 (zero-shot, ontology only) exposes naming and schema deltas; T1 (entity/label enrichment) measurably narrows this gap without retraining; and T2 (lightweight aliasing/regex repairs and harness-guided fixes) delivers most of the remaining gains while keeping the model frozen. This separation between semantic alignment and model training is practical: new sites can be onboarded primarily by ingesting Brick TTLs into Fuseki, setting environment URLs, and curating a small alias/normalization file—rather than re-running end-to-end training. Implications for practice are immediate. Facilities and energy teams gain a conversational layer that speaks the same ontology as their data, returns plots/CSV/JSON artifacts via a simple file server, and integrates with existing SQL/TimescaleDB/Cassandra backends. The microservice contract for analytics is intentionally simple (time–value arrays with optional parameters), enabling incremental addition of new analyses without touching the NLU or knowledge layer. Health endpoints, smoke tests, and a Docker-first deployment reduce operational risk and support repeatable roll-outs across sites with different telemetry stores. Implications for research include a clearer benchmark substrate for NL→SPARQL and multi-step building reasoning. The staged T0–T2 harness makes portability measurable and reproducible, encouraging studies on compositional generalization, alias-robust entity grounding, and schema-aware decoding. The documented dataset schema for NL→SPARQL (v2) and the artifact pipeline (from SPARQL results to analytics outputs and summaries) provide a foundation for releasing richer, reasoning-dependent corpora and for investigating guardrails that constrain LLMs to ontology-consistent narratives.

9 Conclusion

We presented an end-to-end, ontology-driven conversational platform for smart buildings that couples NL→SPARQL reasoning with production analytics and summarisation in a modular, reproducible stack. The system combines Rasa-based NLU, a Brick-backed SPARQL store, a library of time-series analytics exposed via simple HTTP contracts, and optional language services for translation and summarisation. This design yields a traceable pipeline from user queries to executable computations and verifiable artifacts. A staged portability workflow (T0–T2) shows that most cross-building performance can be recovered without retraining by enriching entities and applying lightweight aliasing and schema repairs. The public repository provides per-building Docker stacks, health checks, smoke tests, and a documented artifact flow, allowing others to reproduce results, extend analytics, or evaluate new buildings with low operational overhead. By decoupling semantic alignment from model training and by standardizing analytics interfaces, the framework lowers the barrier to deploying conversational analytics across heterogeneous sites. We expect the accompanying evaluation harness, dataset schema, and open implementation to serve as a practical baseline for future work on compositional reasoning, schema-aware decoding, and trustworthy, privacy-respecting building intelligence.

Appendix A: Extended results

This appendix presents a consolidated set of representative chatbot question-answering examples across all four reasoning classes (C1–C4), demonstrating the system's capability to handle direct factual queries, multi-hop reasoning, temporal aggregations, and complex analytics integration spanning three building deployments.

Table 11 Consolidated Chatbot QA with reasoning classes C1–C4

Index	Class	User-Chatbot Response
1	C1	**User:** Where is LPG_Natural_Gas_Town_MQ5_Gas_Sensor_5.32 located? **Bot:** The LPG_Natural_Gas_Town_MQ5_Gas_Sensor_5.32 is located in the West zone, specifically in Room 5.32.
2	C4	**User:** Can you show me the failure trends for Zone_Air_Humidity_Sensor_5.02 over the last two months, February to March 2025? **Bot:** The analysis shows the failure trends for Zone_Air_Humidity_Sensor_5.02 over the last two months (February to March 2025). The mean value of humidity was 45.2, with a minimum value of 42.0 and a maximum of 48.5. The standard deviation is 2.31, and the latest recorded value is 46.8. The sensor's status indicates that the humidity level is within normal limits. Additionally, the comfort index was calculated to be 92.45, which assesses the overall environment as 'Comfortable'.
3	C4	**User:** What deviations have been observed in Air_Temperature_Sensor_5.08 readings between 01/02/2025 and 15/02/2025? **Bot:** The analysis of Air_Temperature_Sensor_5.08 data between 01/02/2025 and 15/02/2025 shows that the latest reading, which is from 24th April 2025, deviates slightly from the historical mean of 29.014 degrees Celsius by -0.04 degrees Celsius. However, it remains within the normal range as indicated by the "Within normal range." message. The historical standard deviation was 0.025 degrees Celsius.
4	C4	**User:** Can you show the sensor trends for CO_Level_Sensor_5.02 from 01/02/2025 to 28/02/2025? **Bot:** The sensor trend for CO_Level_Sensor_5.02, from 01/02/2025 to 28/02/2025, shows a decrease over the period. The initial rolling mean (average of recent readings) was 29.06 parts per million (ppm), while the latest rolling mean is 28.91 ppm. This represents a slight downward trend, with a difference of approximately -0.15 ppm between the initial and final rolling means.
5	C4	**User:** Can you correlate Air_Temperature_Sensor_5.01 with Air_Temperature_Sensor_5.02 from 01/03/2025 to 15/03/2025? **Bot:** Based on the provided data, it appears that there is a correlation between Air_Temperature_Sensor_5.01 and Air_Temperature_Sensor_5.02 from March 1st to March 15th, 2025. On the date of the analysis (April 25th), the values recorded were 22.14 degrees Celsius for Air_Temperature_Sensor_5.01 and 19.56 degrees Celsius for Air_Temperature_Sensor_5.02. The exact nature of this correlation, such as whether they consistently show similar or opposite trends over the specified period, would require further analysis of the full dataset.
6	C4	**User:** What's the air quality index from 01/02/2025 to 28/02/2025? **Bot:** The air quality index from February 1st to February 28th, 2025, as calculated on April 25th, 2025, is approximately 2.45 with a "healthy" status. The components of the air quality are PM10 at 0.024, NO2 at 0.085, CO at 2.27 (rounded), and CO2 at 0.07425.
7	C4	**User:** Can you aggregate sensor data for Formaldehyde_Level_Sensor_5.01 four hourly basis from 01/01/2025 to 31/01/2025? **Bot:** The request was to aggregate the Formaldehyde Level sensor data for a four-hour interval from January 1st, 2025, to January 31st, 2025. The response provided includes the aggregated mean, standard deviation (std), minimum, and maximum levels of Formaldehyde_Level_Sensor_5.01 for each four-hour interval.

Table 12 Consolidated Chatbot QA with reasoning classes C1–C4

Index	Class	User-Chatbot Response
8	C1	**User:** Where is bldg2.ZONE.AHU01.RM103.Zone_Air_Temp located?
		Bot: It is a zone temperature sensor in Room RM103 served by AHU01 (Building 2).
9	C1	**User:** Which AHU does bldg3.AHU.AHU02S.Outside_Air_Temp belongs to?
		Bot: The sensor belongs to AHU02S in Building 3 (outside air temperature for that AHU).
10	C2	**User:** List rooms under AHU03S that have zone air temperature sensors.
		Bot: The ontology lists multiple rooms under AHU03S with Zone_Air_Temp, including RM300C, RM301A, RM306, RM308, RM309, RM310, RM311, RM312, RM314, and RM315 (non-exhaustive).
11	C2	**User:** Which AHU serves bldg3.ZONE.AHU02S.RM211.Zone_Air_Temp, and what is its supply air temperature point?
		Bot: The room RM211 is served by AHU02S; the corresponding point is bldg3. AHU.AHU02S.Supply_Air_Temp.
12	C3	**User:** What was the average bldg2.AHU.AHU01.Supply_Air_Temp over August 2025?
		Bot: Aggregation (mean) over 2025-08-01 to 2025-08-31 yields an average of 12.8° C (analytics: aggregation.mean).
13	C3	**User:** Provide daily min/max for bldg3.CHW.Building_Chilled_Water_Supply_ Temp in September 2025.
		Bot: Over 2025-09-01 to 2025-09-30, the daily minima and maxima range approximately between 5.8°C and 8.4°C; daily aggregates are available per date (analytics: aggregation.minmax).
14	C3	**User:** Aggregate PM2.5_Sensor.01 readings on a 4-hour basis for June 2025.
		Bot: 4-hour buckets from 2025-06-01 to 2025-06-30 return mean, std, min, and max per interval for PM2.5_Sensor.01 (analytics: aggregation.resample-4h).
15	C4	**User:** Correlate bldg3.AHU.AHU01S.Supply_Air_Temp with bldg3.ZONE. AHU01S.RM111.Zone_Air_Temp from 01/07/2025 to 31/07/2025.
		Bot: Timeseries fusion via UUIDs shows positive correlation (Pearson r $\tilde{0}$.71) with zone lagging supply by 10–20 minutes; pipelines: correlation.pearson and lag scan (analytics: correlation).
16	C4	**User:** Detect anomalies in bldg2.AHU.AHU01.Supply_Air_Pressure between 01/06/2025 and 15/07/2025.
		Bot: Z-score based anomaly detection flags two spikes on 2025-06-18 and 2025-07-02 exceeding 3σ; overall uptime unaffected (analytics: anomaly.zscore).
17	C4	**User:** Compute building air quality index for June 2025 using PM2.5_Sensor.01, CO_Sensor.01, NO2_Level_Sensor.01, and CO2_Level_Sensor.01.
		Bot: Composite AQI over 2025-06-01 to 2025-06-30 is 2.3 ("healthy"); components are normalized and weighted before aggregation (analytics: aqi.compose).
18	C4	**User:** Show weekly trend for bldg3.CHW.LOOP_DP across July 2025.
		Bot: Rolling 7-day mean indicates a $\tilde{5}$% week-over-week decrease in differential pressure; change-points not significant (analytics: trend.rolling).

Acknowledgements This work is supported in part by the EPSRC PETRAS (Grant No. EP/S035362/1), the GCHQ National Resilience Fellowship, and ARCCA at Cardiff University.

Author Contributions All authors contributed equally to this work.

Funding Not applicable.

Data Availability All the datasets are publicly available, and the links are attached at https://github.com/suhasdevmane/OntoBot.

Declarations

Competing interests The authors declare no competing interests.

Ethics approval and consent The research project is approved under the ethics reference COMSC/Ethics/2023/009. Consent is taken when needed from the users.

Consent for publication Authors consent for publications.

References

1. Alavi, H.S., Churchill, E.F., Wiberg, M., Lalanne, D., Dalsgaard, P., Schieck, A.F.G., Rogers, Y.: Introduction to human-building interaction (HBI): Interfacing HCI with architecture and urban design. ACM Trans. Comput. Interact. **26**(2) (2019)
2. Alavi, H.S., Churchill, E., Kirk, D., Nembrini, J., Lalanne, D.: Deconstructing human-building interaction. Interactions **23**(6), 60–62 (2016)
3. Balaji, B., Bhattacharya, A., Fierro, G., Gao, J., Gluck, J., Hong, D., Johansen, A., Koh, J., Ploennigs, J., Agarwal, Y., Berges, M., Culler, D., Gupta, R., Kjærgaard, M.B., Srivastava, M., Whitehouse, K.: Brick: Towards a unified metadata schema for buildings. In: Proceedings of the 3rd ACM Conference on Systems for Energy-Efficient Built Environments, BuildSys 2016, pp. 41–50. Association for Computing Machinery, Inc, (2016)
4. Nageeb, A.A., Mahmoud, A.S., Omar, Y.K.: An approach for building a framework for applying natural language queries on rdf database. In: Hassanien, A.E., Snášel, V., Tang, M., Sung, T.-W., Chang, K.-C. (eds.) Proceedings of the 8th International Conference on Advanced Intelligent Systems and Informatics 2022, pp. 627–638. Springer, Cham (2023)
5. Reif, J., Jeleniewski, T., Gill, M.S., Gehlhoff, F., Fay, A.: Chatbot-based ontology interaction using large language models and domain-specific standards. (2024). https://doi.org/10.1109/ETFA61755.2024.10711065
6. Yao, Y., Huang, L., He, Y., Ma, Z., Xu, X., Mi, H.: Reviewing and reflecting on smart home research from the human-centered perspective. In: Proceedings of the 2023 CHI Conference on Human Factors in Computing Systems. CHI '23. Association for Computing Machinery, New York, NY, USA (2023). https://doi.org/10.1145/3544548.3580842
7. Lecorvé, G., Veyret, M., Brabant, Q., Rojas Barahona, L.M.: SPARQL-to-text question generation for knowledge-based conversational applications. In: Proceedings of the 2nd Conference of the Asia-Pacific Chapter of the Association for Computational Linguistics and the 12th International Joint Conference on NLP (Volume 1: Long Papers), pp. 131–147 (2022)
8. Tran, H., Phan, L., Anibal, J., Nguyen, B.T., Nguyen, T.-S.: SPBERT: An Efficient Pre-training BERT on SPARQL Queries for Question Answering over Knowledge Graphs (2021)
9. Tran, H., Phan, L., Anibal, J., Nguyen, B.T., Nguyen, T.-S.: Spbert: An efficient pre-training bert on sparql queries for question answering over knowledge graphs. (2021)
10. Diomedi, D., Hogan, A.: Entity linking and filling for question answering over knowledge graphs. Technical report (2022)
11. Diomedi, D., Hogan, A.: Question answering over knowledge graphs with neural machine translation and entity linking. (2021)
12. Rony, M.R.A.H., Kumar, U., Teucher, R., Kovriguina, L., Lehmann, J.: Sgpt: A generative approach for sparql query generation from natural language questions. IEEE Access **10**, 70712–70723 (2022). https://doi.org/10.1109/ACCESS.2022.3188714

13. Avila, C.V.S., Vidal, V.M.P., Franco, W., Casanova, M.A.: Experiments with text-to-sparql based on chatgpt. In: Proceedings - IEEE International Conference on Semantic Computing, ICSC, pp. 277–284. Institute of Electrical and Electronics Engineers Inc., (2024). https://doi.org/10.1109/ICSC59802.2024.00050
14. Rangel, J.C., Farias, T.M.D., Sima, A.C., Kobayashi, N.: Sparql generation: an analysis on fine-tuning openllama for question answering over a life science knowledge graph. Technical report (2024). http://ceur-ws.org
15. Xu, S., Liu, S., Culhane, T., Pertseva, E., Wu, M.-H., Semnani, S., Lam, M.: Fine-tuned LLMs know more, hallucinate less with few-shot sequence-to-sequence semantic parsing over Wikidata. In: Bouamor, H., Pino, J., Bali, K. (eds.) Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing, pp. 5778–5791. Association for Computational Linguistics, Singapore (2023). https://doi.org/10.18653/v1/2023.emnlp-main.353 . https://aclanthology.org/2023.emnlp-main.353/
16. Brei, F., Frey, J., Meyer, L.-P.: Leveraging small language models for text2sparql tasks to improve the resilience of ai assistance. (2024)
17. Wang, R., Zhang, Z., Rossetto, L., Ruosch, F., Bernstein, A.: NLQxform: A Language Model-based Question to SPARQL Transformer (2023)
18. Diallo, P.A.K.K., Reyd, S., Zouaq, A.: A comprehensive evaluation of neural sparql query generation from natural language questions. (2023)
19. Lin, J.H., Lu, E.J.L.: Sparql generation with an nmt-based approach. J. Web Eng. **21**, 1471–1490 (2022). https://doi.org/10.13052/jwe1540-9589.2155
20. Yin, X., Gromann, D., Rudolph, S.: Neural machine translating from natural language to sparql. (2019)
21. Green Building XML (gbXML): gbXML. Accessed on 7 Jan 2026
22. Building Energy Data Exchange Specification (BEDES): BEDES. Accessed on 7 Jan 2026
23. Jia, J., Ma, H., Zhang, Z.: Integration of industry foundation classes and ontology: Data, applications, modes, challenges, and opportunities. Buildings **14**(4) (2024)
24. Project Haystack: Project Haystack. Accessed on 7 Jan 2026
25. Borrmann, A., König, M., Koch, C., Beetz, J.: In: Borrmann, A., König, M., Koch, C., Beetz, J. (eds.) Building Information Modeling: Why? What? How?, pp. 1–24. Springer, Cham (2018)
26. ETSI: Smart Applications REFerence Ontology (SAREF) and Extensions. Accessed on 7 Jan 2026
27. Moreno-Munoz, A., Giacomini, N.: Energy Smart Appliances: Applications, Methodologies, and Challenges, 1st edn. Wiley, Newark (2023)
28. Daniele, L., Hartog, F., Roes, J.: Created in close interaction with the industry: The smart appliances reference (saref) ontology. In: Formal Ontologies Meet Industry, pp. 100–112. Springer, Cham (2015)
29. Hwerbi, K.: An ontology-based chatbot for crises management: use case coronavirus. (2020) arXiv:2011.02340
30. Janowicz, K., Rasmussen, M.H., Lefrançois, M., Schneider, G.F., Pauwels, P.: BOT: The building topology ontology of the W3C linked building data group. Semant. Web. **12**(1), 143–161 (2020)
31. World Wide Web Consortium (W3C): Semantic Sensor Network Ontology. Accessed on 7 Jan 2026
32. Haller, A., Janowicz, K., Cox, S.J.D., Lefrançois, M., Taylor, K., Le Phuoc, D., Lieberman, J., García-Castro, R., Atkinson, R., Stadler, C.: The modular SSN ontology: A joint W3C and OGC standard specifying the semantics of sensors, observations, sampling, and actuation. Semant. Web. **10**(1), 9–32 (2018)
33. Balaji, B., Bhattacharya, A., Fierro, G., Gao, J., Gluck, J., Hong, D., Johansen, A., Koh, J., Ploennigs, J., Agarwal, Y., Bergés, M., Culler, D., Gupta, R.K., Kjærgaard, M.B., Srivastava, M., Whitehouse, K.: Brick: Metadata schema for portable smart building applications. Appl. Energy **226**, 1273–1292 (2018). https://doi.org/10.1016/j.apenergy.2018.02.091
34. BuildingSync: BuildingSync. Accessed on 7 Jan 2026
35. Augello, A., Pilato, G., Vassallo, G., Gaglio, S.: In: Gaglio, S., Lo Re, G. (eds.) Chatbots as Interface to Ontologies, pp. 285–299. Springer, Cham (2014)
36. Gallo, S., Paterno, F., Malizia, A.: Conversational interfaces in iot ecosystems: Where we are, what is still missing. In: ACM International Conference Proceeding Series, pp. 273–287. Association for Computing Machinery, (2023). https://doi.org/10.1145/3626705.3627775
37. Bocklisch, T., Faulkner, J., Pawlowski, N., Nichol, A.: Rasa: Open source language understanding and dialogue management. (2017)
38. Mishra, D.S., Agarwal, A., Swathi, B.P., Akshay, K.C.: Natural language query formalization to sparql for querying knowledge bases using rasa. Prog. Artif. Intell. **11**, 193–206 (2022). https://doi.org/10.1007/s13748-021-00271-1
39. Rajosoa, M., Hantach, R., Abbès, S.B., Calvez, P., Calvez, P.: Hybrid question answering system based on natural language processing and sparql query. (2019). https://api.semanticscholar.org/CorpusID:208194622

40. Saini, A., Verma, A., Arora, A., Gupta, C.: Linguistic rule-based ontology-driven chatbot system. In: Bhatia, S.K., Tiwari, S., Mishra, K.K., Trivedi, M.C. (eds.) Advances in Computer Communication and Computational Sciences, pp. 47–57. Springer, Singapore (2019)
41. Lamiae, E., Hicham, G.T., Fatiha, E., Mohammed, B., Hajoub, M.W.: Patient smart home monitoring using vision neural network transformers. In: Proceedings of the 6th International Conference on Networking, Intelligent Systems & Security. NISS '23. Association for Computing Machinery, New York, NY, USA (2023)
42. Lee, T.-H., Kim, H., Lee, D.: Transformer based early classification for real-time human activity recognition in smart homes. In: Proceedings of the 38th ACM/SIGAPP Symposium on Applied Computing. SAC '23, pp. 410–417. Association for Computing Machinery, New York, NY, USA (2023)
43. Huang, X., Zhang, S.: Human activity recognition based on transformer in smart home. In: Proceedings of the 2023 2nd Asia Conference on Algorithms, Computing and Machine Learning. CACML '23, pp. 520–525. Association for Computing Machinery, New York, NY, USA (2023)
44. Wang, M., Yang, N., Weng, N.: Securing a smart home with a transformer-based iot intrusion detection system. Electronics **12**(9) (2023)
45. Gokhale, G., Van Gompel, J., Claessens, B., Develder, C.: Transfer learning in transformer-based demand forecasting for home energy management system. In: Proceedings of the 10th ACM International Conference on Systems for Energy-Efficient Buildings, Cities, and Transportation. BuildSys '23, pp. 458–462. Association for Computing Machinery, New York, NY, USA (2023)
46. Nguyen, H.D., Tran, T.V., Pham, X.T., Huynh, A.T., Do, N.V.: Ontology-based integration of knowledge base for building an intelligent searching chatbot. Sensors Mater. **33**(9), 3101–3123 (2021)
47. Vegesna, A., Jain, P., Porwal, D.: Ontology based Chatbot (For E-commerce Website). Int. J. Comput. Appl. **179**(14), 51–55 (2018)
48. Avila, C.V.S., Calixto, A.B., Rolim, T.V., Franco, W., Venceslau, A.D.P., Vidal, V.M.P., Pequeno, V.M., De Moura, F.F.: Medibot: An ontology based chatbot for Portuguese speakers drug's users. In: ICEIS 2019 - Proc. 21st Int. Conf. Enterp. Inf. Syst. vol. 1, pp. 25–36 (2019)
49. Jung, H., Kim, W.: Automated conversion from natural language query to sparql query. J. Intell. Inf. Syst. **55**, 501–520 (2020). https://doi.org/10.1007/s10844-019-00589-2
50. Ait-Mlouk, A., Jiang, L.: Kbot: A knowledge graph based chatbot for natural language understanding over linked data. IEEE Access **8**, 149220–149230 (2020). https://doi.org/10.1109/ACCESS.2020.3016142
51. Cho, E.: Hey google, can i ask you something in private? In: Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems. CHI '19, pp. 1–9. Association for Computing Machinery, New York, NY, USA (2019)
52. Følstad, A., Araujo, T., Law, E.L.C., Brandtzaeg, P.B., Papadopoulos, S., Reis, L., Baez, M., Laban, G., McAllister, P., Ischen, C., Wald, R., Catania, F., Meyer von Wolff, R., Hobert, S., Luger, E.: Future directions for chatbot research: an interdisciplinary research agenda. Computing **103**(12), 2915–2942 (2021)
53. Fitton, D., Read, J.C., Sim, G., Cassidy, B.: Co-designing voice user interfaces with teenagers in the context of smart homes. In: Proceedings of the 17th ACM Conference on Interaction Design and Children. IDC '18, pp. 55–66. Association for Computing Machinery, New York, NY, USA (2018)
54. Frens, J., Funk, M., Hout, B., Le Blanc, J.: Designing the iot sandbox. In: Proceedings of the 2018 Designing Interactive Systems Conference. DIS '18, pp. 341–354. Association for Computing Machinery, New York, NY, USA (2018)
55. Heck, M., Shon, S.H., Becker, C.: Does using voice authentication in multimodal systems correlate with increased speech interaction during non-critical routine tasks? In: 27th International Conference on Intelligent User Interfaces. IUI '22, pp. 868–877. Association for Computing Machinery, New York, NY, USA (2022)
56. Koshy, V., Park, J.S.S., Cheng, T.-C., Karahalios, K.: "we just use what they give us": Understanding passenger user perspectives in smart homes. In: Proceedings of the 2021 CHI Conference on Human Factors in Computing Systems. CHI '21. Association for Computing Machinery, New York, NY, USA (2021)
57. Louis, J., Rashid, K.: Utilizing building information models as operating systems for smart homes. In: Proceedings of the Workshop on Human-Habitat for Health (H3): Human-Habitat Multimodal Interaction for Promoting Health and Well-Being in the Internet of Things Era. H3 '18. Association for Computing Machinery, New York, NY, USA (2018)
58. Salovaara, A., Bellucci, A., Vianello, A., Jacucci, G.: Programmable smart home toolkits should better address households' social needs. In: Proceedings of the 2021 CHI Conference on Human Factors in Computing Systems. CHI '21. Association for Computing Machinery, New York, NY, USA (2021)
59. Boovaraghavan, S., Patidar, P., Agarwal, Y.: TAO: Context Detection from Daily Activity Patterns Using Temporal Analysis and Ontology. Proc. ACM Interactive, Mobile, Wearable Ubiquitous Technol. **7**(3) (2023)

60. Suhas, D., Rana, O., Lannon, S., Perera, C.: Talking buildings: Interactive human-building smart-bot for smart buildings. In: WISE 2024, December 2–5, 2024, Proceedings, Part I, pp. 399–415. Springer, Berlin, Heidelberg (2024). https://doi.org/10.1007/978-981-96-0579-8_28
61. Jiang, A.Q., Sablayrolles, A., Mensch, A., Bamford, C., Chaplot, D.S., Casas, D., Bressand, F., Lengyel, G., Lample, G., Saulnier, L., Lavaud, L.R., Lachaux, M.-A., Stock, P., Scao, T.L., Lavril, T., Wang, T., Lacroix, T., Sayed, W.E.: Mistral 7B (2023). https://arxiv.org/abs/2310.06825
62. Li, M., Yang, H., Liu, Z., Alam, M.M., Ebrahim, Sack, H., Gesese, G.A.: KGMistral: Towards boosting the performance of large language models for question answering with knowledge graph integration. In: Workshop on Deep Learning and Large Language Models for Knowledge Graphs (2024). https://openreview.net/forum?id=JzL0qm3YA8
63. Gashkov, A., Perevalov, A., Eltsova, M., Both, A.: SPARQL Query Generation with LLMs: Measuring the Impact of Training Data Memorization and Knowledge Injection (2025). https://arxiv.org/abs/2507.13859