BuildingChat: Enabling User Interactions with General-Purpose Sensing Infrastructure for Buildings

PRASOON PATIDAR, Carnegie Mellon University, USA
ANUPAMA SITARAMAN*, Carnegie Mellon University, USA
BEN WEINSHEL*, Carnegie Mellon University, USA
HAOZHE ZHOU*, Carnegie Mellon University, USA
YUVRAJ AGARWAL, Carnegie Mellon University, USA

Smart buildings collect extensive sensor data, but occupants rarely benefit from it: data flows to facility management while occupants navigate fragmented dashboards or lack access entirely. We first conducted an 11-participant formative interview study to understand building occupants' unmet needs, finding that user query needs vary widely. Question-answering systems can support these dynamic use cases, but smart buildings present challenges unaddressed by general-purpose chatbots: buildings contain thousands of sensors generating continuous data, queries often require multi-step analysis across locations, and the user may need guidance to develop queries that do not invade other building occupants' privacy.

Based on these insights, we developed BuildingChat, a conversational system that enables building occupants to query sensor infrastructure directly. BuildingChat decomposes queries into composable template operations, enforces tag-based access control with policy-driven time resolution, and iteratively refines queries when information is missing or access is denied. We show that BuildingChat achieves 88% query accuracy across six language models, outperforming code generation, in-context reasoning, and ReAct agent baselines by 14–46%. Unlike these baselines, which leak restricted data in 13–46% of queries despite policy instructions, BuildingChat enforces access control by construction. We also conducted a three-day usability study with 10 occupants of a building instrumented with over 300 sensors. Participants found BuildingChat easy to learn and its responses understandable, rating it 80±8 on the system usability scale.

CCS Concepts: • **Computer systems organization** → **Real-time systems**; • **Security and privacy** → *Software and application security*; • **Human-centered computing** → **Ubiquitous and mobile computing systems and tools**; **Interactive systems and tools**.

Additional Key Words and Phrases: Ubiquitous Sensing, Interactive Systems, Smart Buildings

ACM Reference Format:
Prasoon Patidar, Anupama Sitaraman, Ben Weinshel, Haozhe Zhou, and Yuvraj Agarwal. 2026. BuildingChat: Enabling User Interactions with General-Purpose Sensing Infrastructure for Buildings. *Proc. ACM Interact. Mob. Wearable Ubiquitous Technol.* 10, 3, Article 145 (September 2026), 40 pages. https://doi.org/10.1145/3831994

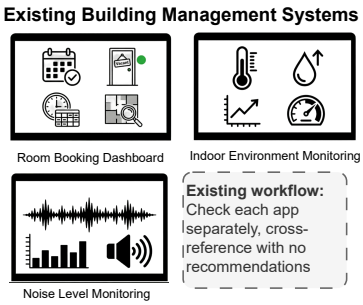
1 INTRODUCTION

Modern buildings are increasingly equipped with sensing infrastructure, from environmental monitors that track temperature, humidity, and air quality to occupancy sensors, energy meters, and custom multi-modality sensing

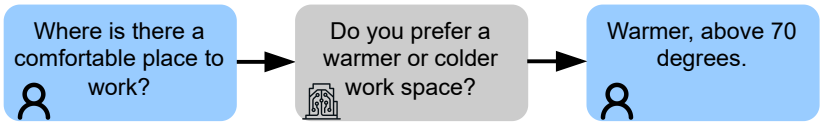
*These authors contributed equally to this research.

Authors' Contact Information: Prasoon Patidar, Carnegie Mellon University, Pittsburgh, USA, prasoonpatidar@cmu.edu; Anupama Sitaraman, Carnegie Mellon University, Pittsburgh, USA, asitaram@andrew.cmu.edu; Ben Weinshel, Carnegie Mellon University, Pittsburgh, USA, bweinshel@cmu.edu; Haozhe Zhou, Carnegie Mellon University, Pittsburgh, USA, haozhezh@cs.cmu.edu; Yuvraj Agarwal, Carnegie Mellon University, Pittsburgh, PA, USA, yuvraj@cs.cmu.edu.

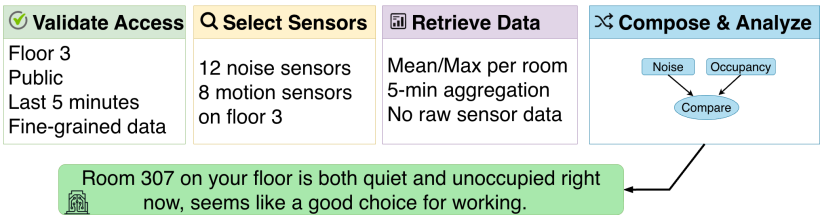
ACM 2474-9567/2026/9-ART145
https://doi.org/10.1145/3831994



(a) To find a comfortable place to work, building occupants today must check multiple applications separately: room booking, noise monitoring, and occupancy dashboards.



(b) BuildingChat provides a conversational interface and can clarify ambiguous requests.



(c) BuildingChat validates access, selects relevant sensors, retrieves aggregated statistics, composes multi-sensor analysis, and generates a contextual recommendation.

Fig. 1. Overview of BuildingChat system

platforms [7]. This proliferation of Internet of Things (IoT) devices has created the potential for buildings to become truly "smart", responsive to occupants' needs, and capable of providing data-driven insights [43]. Despite this technological abundance, a significant gap exists between the potential of smart buildings and their practical utility for everyday users and building managers [16]. Sensor data generated by building infrastructure typically flows to facility management systems, remaining inaccessible to occupants [16]. In commercial buildings, this is partly attributable to industry incentives: Building Management Systems (BMS) vendors focus on facility management and building operation use cases. However, many buildings operate under shared governance structures where operators and occupants have a mutual stake in how spaces function, including universities, hospitals, public institutions, and co-managed workplaces. Buildings also have increasingly rich sensing infrastructure [7] supported by ML pipelines. In these settings, a barrier to occupant access is the absence of mechanisms that can make sensor data available to multiple stakeholders while respecting various privacy constraints.

For building occupants who want to access sensor data from their own offices or public spaces, the available paths are limited. They may use specialized dashboards tied to individual subsystems (*e.g.*, room booking applications or energy displays in common areas), but each requires learning a separate interface. Occupants may also be able to query databases directly which demands technical expertise and careful management of granular access permissions. Prior research has established foundations for deploying sensing at scale [7, 20, 39] and also documented the persistent communication gap between occupants and facility managers [2, 43].

In our work, we show that building occupants find value in being able to describe their needs in subjective, situated natural language. They want to ask whether a space is comfortable, which room is quietest, or why their office feels warm. A conversational interface can meet users at this level of expression without requiring knowledge of what sensors exist or how the sensor data is organized. We present BuildingChat (Figure 1), a system that enables building occupants to query[1] sensor data through natural language conversation. Rather than requiring users to navigate specialized dashboards or formulate database queries, BuildingChat provides a single entry point for accessing multi-modality sensor data (temperature, humidity, motion, thermal, audio, light,

[1]Throughout this paper, a *query* refers to a natural language question posed by a building occupant to BuildingChat, such as "Is my office warm?" or "Which floor is quietest?" When we discuss the structured representations derived from these questions, we refer to them explicitly as structured or database queries.

vibration, *etc.*) across the building. To understand what building occupants may want to ask from a conversational interface, we first conducted a formative study with 11 participants with five roles: private office occupants, resource managers, building managers, shared space users, and emergency personnel. We found that query needs differed by role, and included personal comfort questions and cross-location comparisons. Furthermore, we found that participants raised concerns about surveillance and the potential for queries to reveal potentially identifiable patterns about individuals.

These findings point to challenges beyond simple question-answering. The query diversity participants described, from point measurements to cross-location comparisons to temporal patterns, means that answers often require multi-step analysis: identifying relevant sensors, aggregating readings by location, and comparing results. Two approaches might seem promising to address this: (1) providing sensor data directly to Large Language Models (LLMs) for reasoning, or (2) having an LLM generate retrieval and analysis code for each query. However, buildings contain thousands of sensors generating continuous time-series data, which is too much to fit in an LLM's context window. Code generation offers flexibility but sacrifices explainability and predictability of responses. The surveillance concerns participants raised impose further constraints. In many building deployments, sending raw sensor data to cloud-based LLMs raises privacy concerns. Since users may not fully understand what data exists and the access control policies, a system should guide users to refine their query towards what the system can answer. BuildingChat addresses these challenges through three mechanisms. First, BuildingChat decomposes queries into composable template operations that execute over pre-aggregated sensor statistics, enabling multi-step analysis while maintaining predictable, explainable execution that works with locally deployed LLMs. Second, when a query is underspecified or involves inaccessible data, the system generates clarifying questions or proposes alternative formulations, proceeding only with explicit user consent. Third, BuildingChat enforces data access policies that constrain queries across multiple dimensions, including spatial scope, temporal resolution, sensor count, and query rate, so that combinations of allowed queries cannot be used to reconstruct individual patterns.

We evaluate BuildingChat through three complementary studies. First, we measure end-to-end response accuracy, the fraction of queries whose final answers match programmatically computed ground truth, on a mock building with controlled data, comparing against three baselines representing established LLM system paradigms: code generation, in-context data reasoning, and a ReAct agent loop. BuildingChat achieves 88% accuracy, outperforming the ReAct agent (74%), in-context reasoning (67%), and code generation (42%), and maintains the highest accuracy across all six models spanning local and cloud deployments. Second, we evaluate whether alternative designs can enforce equivalent access control. When baselines are augmented with explicit policy descriptions and instructed to reject unauthorized queries, they still leak restricted data in 13–46% of cases, with the ReAct agent exhibiting the highest leak rate despite being the most accurate baseline; BuildingChat does not leak data as access control is enforced at the building management layer. Third, we conduct a usability study with 10 participants in a building instrumented with over 300 multi-modality sensors. Participants found BuildingChat easy to learn and its responses understandable, while also surfacing open challenges in interpreting sensor values and reliability for LLM-based systems. In summary, we make the following contributions:

(1) Through an 11-participant formative study, we identified that smart building system needs vary depending on user role and that tensions exist between desired functionality and privacy.

(2) We developed BuildingChat, a conversational interface for building sensor infrastructure that decomposes queries into composable template operations, enforces tag-based access controls, and iteratively refines queries when information is missing or inaccessible.

(3) We demonstrate that BuildingChat reaches 88% accuracy on objective evaluation, exceeding baselines by 14–46%, and prevents the unauthorized data leaks present in 13–46% of baseline queries. Through a three-day deployment with 10 building occupants, we show it yields a SUS score of 80 ± 8, with all participants finding responses understandable. We also identify open challenges in interpreting sensor values and LLM reliability.

2 RELATED WORK

We position our work within four prior research themes: smart building infrastructure, natural language interfaces for IoT systems, LLM-based query or code generation, and access control for conversational and building data systems.

2.1 Smart Building Infrastructure

In commercial buildings, building management systems (BMS) play a central role in managing resources including lighting, heating/cooling, and energy management. BMSs are fragmented and underutilized; Hahn et al. found that BMSs are primarily used for a handful of cases such as energy management, managing occupant comfort and fault detection and diagnosis [16]. To address fragmentation, standardized schemas such as Brick have been developed to enable application portability [4, 8, 12]. Existing BMS deployments generally involve building-wide deployment of sensors (*e.g.*, temperature, motion), but do not typically make granular data easily accessible to building occupants. In response, deployments such as MakeSense [20], SmartSantander [39] and Mites [7] have explored deploying additional sensors, with rich sensing modalities and higher sampling rates.

2.2 Natural Language Interfaces for IoT Systems

Natural language interfaces provide a familiar interface and lower the interaction barrier with complex systems. In the smart home domain, systems such as Sasha [22] and SAGE [37] use LLM-based agents that respond to underspecified user commands like "make it cozy," decompose goals into action plans, and dynamically construct prompts. Other systems focus on enabling lower-power on-device reasoning [23]. While smart-home-focused systems focus primarily on real-time querying and control, systems such as DeepSQA [45] and SensorChat [48] focus on providing a natural-language interface to historical sensor data in personal sensing contexts such as wearables and fitness trackers where a single user owns all data. For buildings, BuildingGPT [26] proposed retrieval-augmented generation (RAG) for querying building metadata in Brick Schema, enabling querying what sensors and equipment exist but not analytical queries over their readings. BuildingSage [13] introduced an AI copilot that generates code for querying Brick metadata and sensor data, addressing data exfiltration risks by keeping raw data on-premise and sharing only the metadata schema with cloud LLMs. In shared buildings with multiple occupant types, however, privacy risks also arise from occupants themselves querying data about each other's spaces in ways that may be privacy invasive. This internal threat model, where the same query may be appropriate from one stakeholder but invasive from another, is complementary to data locality and is the focus of BuildingChat's access control framework.

2.3 LLM-Based Query and Code Generation

LLM-based approaches to data analysis differ in what the model produces and what executes it. In query generation, the LLM translates natural language into a declarative query, such as SQL or SPARQL, that a trusted engine executes, so the output space is constrained by the query language and execution is bounded by the engine's semantics [35, 41]. This direction is studied extensively as knowledge graph question answering (KGQA) and text-to-SPARQL generation, both before and since the advent of LLMs [24, 38]. In code generation, the LLM instead synthesizes executable programs [10], offering flexibility beyond any fixed query language at the cost of behavior that cannot be inspected or audited before execution. Other paradigms include API-calling agents that chain tool invocations [30, 36] and systems that reason over data placed directly in the LLM's context [49].

These paradigms have been applied to building data. Within the KGQA lineage, BuildingQA [28] contributes a benchmark for building knowledge graph question answering and evaluates generation methods spanning zero-shot prompting and agentic generate-and-critique loops, while BIM-GPT [52] enables natural language access to BIM models; both address metadata discovery rather than queries over sensor readings. In companion work,

Mulayim et al. [29] extend natural language access to the readings themselves, chaining text-to-SPARQL retrieval of time series references with LLM-generated Python code that extracts and analyzes the corresponding data. The authors identify ambiguous queries as a key limitation and call for interactive clarification mechanisms of the kind BuildingChat's reformulation provides. OntoSage [14] combines Brick Schema-based SPARQL generation with 30 analytics microservices over time-series sensor data; each query is routed to a single analytics function, so queries requiring multiple analytical operations cannot be expressed, a limitation that motivated our composable template approach (Section 4.3). JARVIS [25] similarly bridges query understanding and sensor analytics for HVAC systems through a two-stage LLM framework for SQL-based retrieval and statistical processing. These systems demonstrate that LLM-mediated access to building sensor data is feasible, but operate in single-user settings where one operator interacts with the full sensor inventory. Building sensor data also challenges query generation more broadly: time-series readings require temporal aggregation rather than point lookups, spatial hierarchies require context-aware resolution, and occupants routinely underspecify queries with implicit context (*e.g.*, "my office feels warm"). Handling these diverse and complex query types safely and in a privacy-preserving manner is the focus of our system design.

2.4 Access Control in Conversational and Building Data Systems

Conversational systems pose a distinct challenge for access control: the language model interpreting requests is non-deterministic, while policy enforcement must be deterministic and auditable. Planas et al. integrated role based access control (RBAC) into chatbot specifications through a domain-specific language, supporting both restricting which queries a role can trigger and varying response precision levels based on user privileges [33, 34]. In their framework, precision tiers are fixed per role; building sensor queries introduce additional complexity as the appropriate resolution depends on query attributes such as recency and spatial scope, not role alone. Other works (*e.g.*, AirGapAgent [3] and LLMAC [54]) use an LLM or other ML models to make access decisions; however, LLMs are non-deterministic and may be vulnerable to prompt injection; in BuildingChat, we use LLMs to help compose a query compatible with the access control policy, but do not use an LLM to enforce the policy. SEAgent [18] applied attribute-based access control to prevent privilege escalation in LLM agent tool use, but assumed resources are discrete (intents, tools, documents). When multiple stakeholders query shared continuous sensor data, access decisions must also account for query parameters.

In the building and IoT domain, privacy research has primarily studied smart home contexts with single users or small households [31, 32]. SmartAuth [42] proposed user-centered authorization for IoT applications, and work on interactive privacy negotiation [53] demonstrated that groups in multi-occupant scenarios can reach privacy consensus through structured interfaces. Zhang et al. [50] found that users of LLM-based conversational agents face trade-offs between privacy, utility, and convenience with limited awareness of disclosure risks, suggesting that conversational building systems must carefully manage what information is revealed through query responses. Researchers have also explored differential privacy for building data [21], though achieving meaningful guarantees for aggregate location data remains challenging in practice [17]. Some work has explored how to gracefully handle scenarios where a user's query violates an access control policy. Adeniye et al. [1] proposed a system that guides users toward privacy-compliant alternatives when queries cannot be fulfilled, similar to BuildingChat's query reformulation mechanism, though it operates on static graph data rather than continuous sensor streams. In the ontology domain, query relaxation techniques systematically broaden failing SPARQL queries to return partial results [5], though these operate silently on structured queries rather than negotiating alternatives with users through natural language.

Table 1. Comparison across capabilities required for conversational sensor data access in multi-stakeholder buildings. Rows 1–6 are building data systems; rows 7–8 are access control frameworks for conversational systems, for which sensor analytics and query composition are outside their design scope (—). ✗ indicates the capability is absent. We compare the key related works in each category.

#	System	Sensor/Metadata Analytics	Multi-Criteria Composition	Multi-Stakeholder Access Control	Query Reformulation
1	BuildingGPT [26]	✓ metadata only	✗	✗	✗
2	BuildingQA methods [28]	✓ metadata only (zero-shot, agentic SPARQL)	✗	✗	✗
3	Mulayim et al. [29]	✓ SPARQL + LLM generated code	✗	✗	✗
4	BuildingSage [13]	✓ LLM generated code	✗	✗	✗
5	OntoSage [14]	✓ 30 specialized functions	✗	✗	✗
6	JARVIS [25]	✓ LLM + SQL retrieval	✗	✗	✗
7	Planas et al. [33, 34]	—	—	✓ role-driven (programmatic)	✗
8	LLMAC [54]	—	—	✓ role-driven (LLM-based)	✗
	BuildingChat	✓ *general-purpose*	✓ *Composable operators*	✓ *role+query driven (programmatic)*	✓ *consent-based alternatives*

2.5 Research Gap

Smart buildings consist of multiple stakeholders (*e.g.*, occupants, resource managers, facility staff, emergency responders) who share physical space and may have legitimate but conflicting interests in the same sensor data. No single user owns the data, and the same query may be appropriate from one person but constitute surveillance from another. They are distinct from traditional databases in that users often do not know what data exists, making simple access denials uninformative and frustrating. The difficulty lies not in missing components but in their integration (Table 1). Prior work chains query generation, data extraction, and analysis end to end for a single operator [29]. In shared buildings, however, whether a query is permitted depends on its parameters: the same user may be allowed a reading at one temporal resolution but not another, and data accessible in aggregate may be restricted for individual spaces. Enforcing such policies, explaining denials, and proposing accessible alternatives all require these parameters to be visible before any data is retrieved, and generated code and free-form queries do not expose them for policy evaluation and negotiation. This motivated the design of BuildingChat, where query understanding, analysis, and access control operate over a shared structure of composable, inspectable operations, allowing policies to be evaluated and alternatives negotiated before execution.

3 SOLICITING USER NEEDS FROM SMART BUILDINGS

We conducted a semi-structured interview study with 11 participants to understand how different building stakeholders interact with their spaces and what they would want from a smart building system. We designed

the study to address two research questions: **RQ1:** How would different personas interact with smart building systems via a chatbot interface? **RQ2:** What are concerns that users have around the deployment and use of smart building chatbot systems?

3.1 Method

We targeted five personas for our study: *private office occupants*, *shared office occupants*, *resource managers* who manage scheduling and IT resources, *building managers* who manage building systems, and *emergency managers* responsible for building safety. Participants were required to be 18 years or older and fluent in English. We recruited 11 participants: three private office occupants, three shared office occupants, two resource managers, one building manager, and two emergency managers.

Our semi-structured interviews lasted 60 minutes and were conducted either in-person or remotely; participants were paid $15 USD. The protocol consisted of four phases: (1) establishing participants' relationship with their building, (2) introducing a hypothetical smart building with sensing capabilities (temperature, sound levels, motion) to ground the discussion, (3) exploring desired queries and interactions, and (4) discussing privacy concerns and boundaries. Interview recordings were transcribed and analyzed using thematic analysis by one researcher to identify emerging themes. The study was approved by our institution's IRB.

Limitations: The sample (n=11) was drawn from a single institution, and participants responded to a hypothetical system rather than an actual deployment. Therefore, it is possible that the insights we found are catered towards a specific community. In addition, the five roles we studied may not capture all building stakeholders, such as visitors or maintenance staff such as janitors.

3.2 Findings

3.2.1 Query needs vary by role and information granularity. Participants' information needs were clustered into three groups that differed in topic, granularity and completeness of information they require (Table 2). Building occupants with private or shared offices oriented their queries around personal comfort and resource availability. Faculty participants wanted to understand conditions in their own offices, *e.g.*, temperature (P10), interruption frequency (P6), and whether collaborators were nearby (P4, P6). Students focused on finding spaces to work (P3, P8, P11), asking which rooms were quiet (P3, P8) and whether shared resources like microwaves were currently available (P3, P8). Both resource managers (P7, P9) described how a system could help identify problems with spaces before they become disruptive to others in the building. However, they worried about information overload and wanted actionable summaries rather than raw data. One of the resource managers specifically requested temporal visualization: "that would really help to be able to see these kind of things graphically on a timeline for a space or a building." Emergency managers had distinct needs centered on crisis response, *e.g.*, detecting gas leaks, locating occupants during emergencies, and identifying escape routes. They wanted maximum information completeness and expressed no concern about overload. Given these findings and the broad set of use cases that varied between personas, we thought a chatbot design would be appropriate for BuildingChat to provide flexibility and support diverse use cases.

3.2.2 Queries require multi-step reasoning. Beyond simple sensor lookups, many of the questions participants described required reasoning across multiple locations or time periods. Of the 80 collected queries answerable from the building's sensor data, 58 (72%) require multi-step analysis by either retrieving and comparing data across multiple locations or analyzing temporal patterns (Table 2). For example, to answer P6's question "Is it that all offices are this loud, or this hot, or is it just that I'm the unlucky one today?", a query system would need to (a) retrieve temperature and audio for P6's office, (b) identify other offices on the same floor, (c) retrieve and aggregate readings from those offices, and (d) compare P6's values against the aggregate. No single database lookup can produce this answer. Recent work on tabular and time-series reasoning confirms that decomposing

Table 2. Query distribution from our formative study by role and analytical complexity. Point measurements retrieve a single sensor reading; cross-location queries compare or filter across multiple sensor groups; temporal queries analyze patterns over time; outside scope queries require capabilities beyond sensing (*e.g.*, wayfinding).

Role	Participants	Point Measurement	Cross-Location (Compare/Filter)	Temporal Analysis	Outside Scope	Total
Emergency Managers	2	8	3	0	6	17
Shared Space Users	3	5	17	3	9	34
Office Occupants	3	6	7	3	6	22
Resource Managers	2	1	3	8	2	14
Building Manager	1	2	8	6	4	20
Total	**11**	**22 (28%)**	**38 (47%)**	**20 (25%)**	**27**	**107**

multi-step analytical queries into structured operations with deterministic execution outperforms end-to-end LLM reasoning [19, 47]. Beyond comparison, questions about space availability require checking the status of multiple rooms and filtering based on current occupancy. Temporal patterns represented another common category: two participants (P5, P9) wanted to understand when spaces were most heavily used, which requires aggregating occupancy data over time rather than reporting instantaneous readings.

3.2.3 Privacy concerns center on surveillance and inference. When asked about concerns with a smart building system, participants' responses diverged by role. Emergency managers engaged minimally with privacy questions, focusing instead on information accuracy. Occupants expressed substantial surveillance concerns. Five of eleven participants explicitly worried about authorities using such a system for surveillance. P10 emphasized the need for prevention of "behavior patterns of the occupants in an identifiable way." Two participants expressed skepticism that meaningful privacy was achievable with such systems. This created direct tension with desired functionality. Faculty participants wanted presence information for everyday coordination, such as whether collaborators were on campus or students were around to meet. But students and staff worried about exactly this capability being used against them. One participant (P4) acknowledged this tension, indicating they would prefer data to be available but would protect themselves in ways they felt necessary to maintain their privacy. Several participants requested specific mechanisms: opt-in consent (P8, P10), need-to-know access restrictions (P8, P9), and aggregate-only data without individual-level detail (P3, P9).

3.3 Design Implications

These findings shaped BuildingChat's design in three ways. (1) The diversity of query types, spanning point-in-time measurements, temporal patterns, and cross-location comparisons, combined with the multi-step reasoning many queries require, motivated a semi-structured approach where common query patterns are encoded as pre-defined operations that can be composed together to answer complex questions. (2) The surveillance concerns raised reveal that access control for building sensors differs fundamentally from traditional database permissions: privacy risk emerges from the interaction of multiple dimensions, including the person asking the query, the spatial and temporal granularity, and the accumulation of queries over time to reveal patterns. (3) Participants frequently expressed queries that were underspecified, using vague qualifiers, contextual references like "my office," and implicit temporal context (*e.g.*, "has my office been noisy lately?"), while simultaneously expressing concern about being denied access without recourse. These patterns suggest that a conversational interface cannot simply reject problematic queries but must guide users toward alternatives that are both answerable and privacy-respecting.

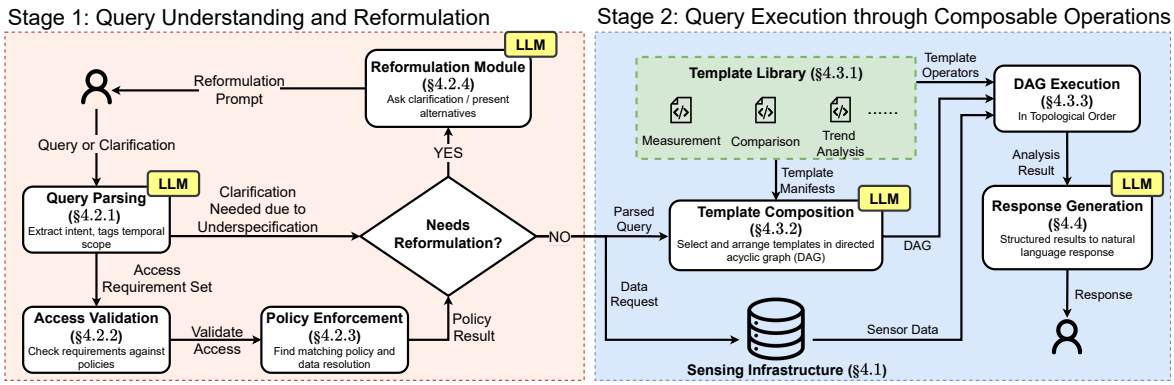


Fig. 2. System overview for BuildingChat. **Stage 1 (Query Understanding and Reformulation):** User queries are parsed to extract intent, spatial tags, and temporal scope (§4.2.1). The system derives access requirements and validates them against policies (§4.2.2). If the query is underspecified or unauthorized, the reformulation module (§4.2.3) proposes default interpretations or accessible alternatives, prompting the user for confirmation or clarification. **Stage 2 (Query Execution):** Once validated, the template composition module selects and arranges templates (§4.3.1) from the library into a Directed Acyclic Graph (DAG) (§4.3.2). The system retrieves aggregated sensor statistics based on the approved policy, executes template operators (§4.3.3), and generates a natural language response (§4.4). Yellow badges indicate LLM-powered components; other components execute deterministic logic.

4 SYSTEM DESIGN

Figure 2 presents an overview of BuildingChat. Consider a user who asks *"Where is there a comfortable place to work?"*. During the first stage, BuildingChat parses this query to extract intent, spatial scope, and temporal scope (§4.2.1), then validates access requirements against data access policies (§4.2.2). Since "comfortable" is ambiguous, the reformulation module (§4.2.3) checks the user's profile preferences and, finding no comfort preference set, proposes a default interpretation prioritizing noise and lighting. The user confirms or adjusts, and we move to Stage 2. First, we retrieve sensor data aggregated to the permitted resolution, and in parallel compose a Directed Acyclic Graph (DAG) of template operations (§4.3.2): parallel branches process noise and light data for candidate spaces, then merge at a comparison node that ranks spaces by both criteria. Templates execute deterministically (§4.3.3), and the response generator (§4.4) produces a natural language recommendation identifying the best matching spaces, accompanied by visualizations comparing the requested metrics over candidate locations.

This two-stage architecture addresses three interconnected challenges from our formative study. The prevalence of multi-step queries (72% of in-scope queries, Table 2) requires a composition mechanism that chains analytical operations rather than treating each query as a single lookup. The surveillance and inference concerns participants raised (Section 3.2.3) require that access control be enforced before any data is retrieved, so that denied queries can enter reformulation rather than failing silently or leaking unauthorized data. And the frequency of underspecified queries requires the system to negotiate with users before committing to execution. These challenges are entangled rather than independent. Access validation must precede composition because the set of permitted data determines which template arrangements are valid, and reformulation must precede both because clarifying the query may change what data is requested. This ordering ensures that by the time a query reaches Stage 2, it is fully specified, authorized, and user-approved.

Table 3. An illustration of tag types and values for sensor organization.

Tag Type	Example Values	Privacy Relevance
visibility	public, shared, individual	Primary access dimension
floor	1, 2, 3, 4	Spatial granularity
cardinality	NE, NW, SE, SW	Building quadrant
locationType	office, corridor, study room	Space classification
owner	user email	Personal space access
department	organizational units	Group-level access

4.1 Smart Building Infrastructure

BuildingChat is designed to integrate with smart building deployments that have sensors throughout the building and store historical sensor data. In our implementation, we integrated it with BuildingDepot [44], an open-source building operating system, deployed in an academic building with sensors in offices, meeting rooms, common spaces, and hallways. The data collected from these sensors include audio levels (microphone), vibrations (accelerometer), motion (PIR), low resolution (8x8) thermal imaging, temperature, humidity, and light levels. Sensor metadata is defined using normalized *tags* and *attributes*, which indicate metadata such as the *location type* (*e.g.*, office or common area), *location* (*e.g.*, room 345, 3rd floor northeast), and the *owner/occupants* (relevant tags are shown in Table 3).

To implement BuildingChat, we extended BuildingDepot with a new API to query *aggregate* sensor data. As our formative study revealed, many queries share a common structure: they involve (1) identifying relevant sensors, (2) grouping or aggregating readings, and then (3) filtering or ranking results. To support the access control model discussed in Section 4.2.2 and to ensure efficient database queries, we implemented the first two steps within the BuildingDepot codebase. Existing BMS platforms aggregate sensor data for pre-configured facility dashboards and gate visibility through coarse role-based tiers (administrator, operator, viewer). They do not support per-query access policies that couple temporal resolution with spatial granularity, nor do they expose interfaces for occupants to pose ad-hoc analytical queries. Our aggregate API addresses both gaps: it computes aggregations on demand based on query parameters, and it enforces access policies that constrain what resolution is permitted for each request. A query consists of a list of tag values and sensing modalities (*e.g.*, common areas and meeting rooms), time window (*e.g.*, last 12 hours), aggregation window (*e.g.*, 1 hour) and aggregation function (*e.g.*, median). We discuss the aggregate data query system further in Section 4.2.2.

4.2 Query Understanding and Reformulation

The first stage of BuildingChat transforms natural language queries into structured representations and negotiates with users when queries cannot be fulfilled as requested. This section describes how the system handles both unauthorized and underspecified queries, and the reformulation mechanism that guides users toward accessible alternatives.

4.2.1 Three-Stage Query Parsing. Based on our formative study, we identified five query components required for answering building-related queries: intent, sensor modalities, spatial constraints (expressed through tags), people, temporal scope, and qualifiers (Table 4). We separate parsing into three stages, each implemented as a separate LLM call (complete prompts in Appendix A). The first stage identifies which tag *types* are relevant to the query: the LLM receives the user's query along with the building's tag taxonomy (the set of available tag types such as `floor`, `locationType`, `owner`) and outputs which tag types are referenced. For our running example, *"Where is there a comfortable place to work?"*, the first stage identifies that `locationType` and `location`

Table 4. Query structure components extracted during parsing.

Component	Description	Example
Intent	Information need	comparison, measurement, trend
Metrics	Sensor modalities	temperature, noise, occupancy
Locations	Spatial scope via tags	floor=3, locationType=study_room
Temporal	Time range	"right now", "today", "last week"
Qualifiers	Subjective descriptors	"warm", "quiet", "busy"

are relevant (to find and enumerate candidate workspaces). The second stage performs full structured extraction: given the query, the relevant tag types identified in stage one, and the user's profile context, the LLM extracts specific values, intent, metrics, and temporal range. The user's profile context includes their office assignment and floor (retrieved from BuildingDepot), their timezone, and any preferences the user has configured through the profile settings. This context enables the parser to resolve spatial self-references deterministically: "my office" maps to the user's assigned room tags, "my floor" to their floor tag, and "here" to their current building context, without requiring clarification. When users ask about "occupancy" or "how busy" a space is, the system maps this to the relevant sensor modalities (motion, thermal imaging, audio levels, light) rather than expecting users to name specific sensor types. For this query, the second stage extracts an intent of location recommendation, metrics of noise and light (after clarification resolves "comfortable" to quiet and well-lit), and defaults temporal scope to last minute. The system also checks whether the query would be allowed under the access control policy (Section 4.2.2) before querying for sensor data to allow the query to be reformulated if necessary. The final stage extracts relevant qualifiers for the query that describe the location and sensor modalities (e.g., subjective terms like "warm" or "good", and vague location terms such as "here" or "there"). In this stage, we ask the LLM to serve as a judge to determine whether the qualifier needs additional clarification from the user. This separation into multiple LLM calls addresses a challenge we encountered during development: single-pass parsing confused tag types with tag values as the taxonomy grew, and qualifier judgments were more accurate when isolated from other extraction tasks.

Our formative study revealed that users expect a conversational system to understand their subjective context, and routinely underspecify queries: participants used phrases like "my office," "here," or "somewhere quiet" without providing explicit spatial coordinates, omitted temporal scope when asking about current conditions, and used qualitative descriptors like "warm" or "busy" without defining thresholds (See Figure 3 Example 2). BuildingChat resolves spatial self-references through the user's profile context during parsing, as described above. For ambiguous qualifiers, the system first checks the user's configured preferences: if a user who has set "temperature" and "noise" as their comfort priorities asks *"Where is there a comfortable place to work?"*, the system resolves "comfortable" to those metrics without prompting. When qualifiers cannot be resolved through preferences, the system proposes a concrete default interpretation for the user to accept or reject, rather than asking open-ended clarifying questions. Once parsing produces a complete structured query, either directly or after user confirmation, the system validates whether the user has access to the requested data.

4.2.2 Access Control Framework. In a building context, occupants have varying privacy expectations; in our formative study, many participants expressed concerns about sensor systems being used for monitoring and surveillance (Section 3.2.3). The community norms and policies for each deployment likely differ, but we believe sensor systems should be designed to aid in understanding the building and *not* support tracking individuals. Traditional role-based access control approaches are inadequate in this respect: they assign permissions too coarsely, so if facility managers are allowed to access granular motion data to understand usage trends, they

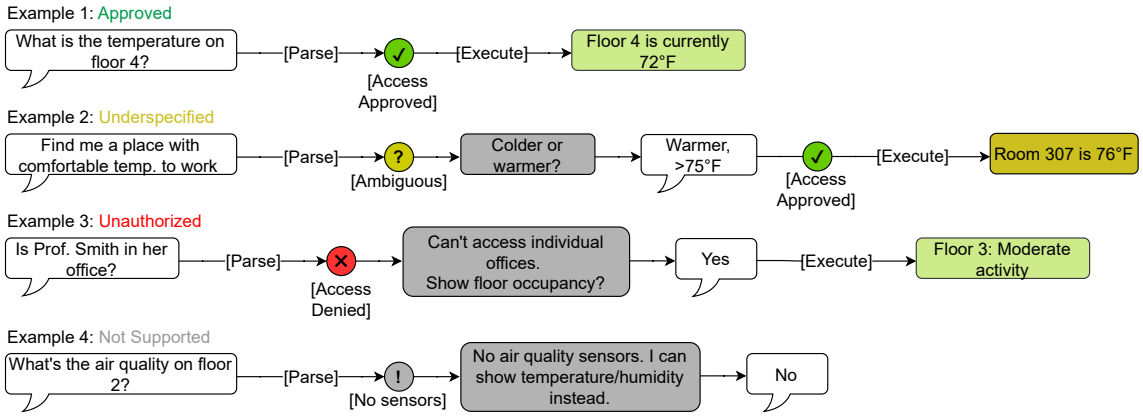


Fig. 3. Query processing traces illustrating four scenarios: (1) a well-specified query proceeds directly to execution, (2) an underspecified query is resolved through profile preferences or a proposed default, (3) an unauthorized query prompts the system to propose an accessible alternative requiring user consent, and (4) a query beyond the building's sensing capabilities ends with an explanation of available alternatives.

would likely also have the ability to see when an individual is using their office. In some cases, however, it may be acceptable for *aggregate* data to be queried based on defined policies. For example, while allowing building managers to access motion data for an individual office would be problematic, a system where they can learn motion trends aggregated across an entire building floor may be acceptable.

As discussed in Section 4.1, queries to BuildingDepot include a list of metadata tags to match, start and end time, aggregation time window, and aggregation function. Each query is checked against the list of *data access policies* and allowed only if it is permitted under at least one policy. The policies include:

- *Required metadata tags:* List of filters that must be included in this policy, *e.g.*, `owner: <user_id>`.
- *Optional metadata tags:* List of optional tags to further filter sensors, *e.g.*, `floor`. Unspecified tags are not permitted to prevent excessively narrow queries.
- *Minimum number of sensors:* Integer. The query is only permitted if it matches at least this many sensors.
- *Aggregation:* List of `[recency, time resolution]` tuples that define the minimum aggregation window for a query about data with a given recency, *e.g.*, `[last day, 15 minute granularity], [last week, 3 hour granularity]`. This policy allows for querying more granular recent data, while limiting the ability to query detailed data about past activity (when combined with a rate limit policy below).
- *Rate limits:* List of `[limit, time frame]` tuples that define the maximum number of queries in a rolling time window, *e.g.*, [10 queries, last day].

This policy framework is designed to support various types of queries, including accessing data from the building occupant's own office (higher time granularity, not rate limited), and querying aggregate values across locations including other people's private offices (higher minimum number of sensors, lower time granularity, lower rate limits). Table 5 shows the policies used in our deployment designed based on feedback from the formative study and community feedback from the deployment of the sensor system; the policies used for each deployment will likely vary based on community norms and expectations. Queries that fail validation against these policies enter the reformulation module, which proposes accessible alternatives rather than simply denying the request.

Table 5. Data access policies used in our deployment

(a) We included policies for personal data access, querying information about public spaces, and querying aggregates across larger sections of the building (which may include private offices). The `location` tag refers to the room number, `floor` and `cardinality` to approximate location in the building, and `visibility` and `locationType` to the type of space (*e.g.*, public corridor).

Use Case	Required Tags	Optional Tags	Min. Sensors	Time
Accessing own office	owner=<user_id>	location	1	unrestricted
Checking public spaces	visibility=public	location, floor, cardinality	1	Table 5b
Multiple-room aggregates	floor	locationType, cardinality	14[a]	Table 5b

[a]To protect individual privacy by providing k-anonymity and address community feedback, we only allow queries including other people's private offices if at least seven offices are included (with two sensors in each office).

(b) We allow high-granularity access to more-recent data, and lower-granularity access to historical data

Recency	Time Resolution	Effect
15 minutes	5 seconds	Current state, fine resolution
1 hour	1 minute	Recent history, moderate aggregation
1 week	1 hour	Historical, coarse aggregation

4.2.3 Reformulation Module. When a query cannot proceed as requested, either because the user lacks access to the requested data or because the query falls outside the system's sensing capabilities, the reformulation module proposes a concrete alternative rather than simply denying the request (See Figure 3). For access denials, the module receives the original query and the reason for denial, then generates an alternative that addresses a similar information need using data the user is authorized to access. For example, if a user requests noise data for individual offices on their floor, the system explains the access restriction and proposes checking noise levels in public areas on the same floor instead (Figure 5). For queries beyond the system's capabilities, the module explains what sensing modalities are available and suggests the closest answerable query. In both cases, the system presents the proposed alternative with explicit accept and reject options, proceeding only with user consent. If the user rejects the proposed alternative, they can reformulate their query manually or end the interaction. BuildingChat maintains session state to avoid re-proposing alternatives the user has already rejected. Intent clarification in LLM-based systems is an active research area with open challenges around generalization across domains and query types [27, 51]. Our clarification and reformulation module is tailored to the building sensor domain, handling the specific forms of underspecification we observed in our formative study (vague qualifiers, implicit spatial context, ambiguous temporal scope) rather than providing a general-purpose solution. Once a query is fully specified and authorized, it proceeds to template composition for execution.

4.3 Query Execution through Composable Operations

When all access requirements for a query are approved, the database handler retrieves sensor data for each validated requirement. This data arrives as an *input packet*: a structured container holding the tag combination that defined the sensor group, the column metadata describing each feature, and the time-indexed values. We adopt a compositional approach where common analysis patterns are encoded as reusable operations called *templates*. Each template performs one well-defined function. Many queries from our formative study require retrieving data from independent sources and merging the results: a comparison retrieves measurements from two

Table 6. Packet types defining data contracts between templates.

Packet Type	Contents	Typical Producer
input_packet	Tag-aggregated sensor statistics	Database handler
measurement	Single value with location and metadata	current_measurement
measurement_set	Collection of related measurements	Parallel measurement branches
analysis_result	Findings, comparisons, or diagnostics	spatial_compare, temporal_analysis
location_set	Locations meeting filter criteria	threshold_filter
time_series	Temporal sequence of values	temporal_analysis

Table 7. Template library derived from our formative study query patterns.

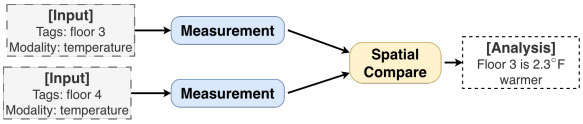
Template	Inputs	Outputs	Formative Pattern
current_measurement	input_packet	measurement	Point-in-time comfort queries
spatial_compare	measurement+	analysis_result	Cross-location comparisons
temporal_analysis	input_packet	time_series, analysis_result	Pattern and trend detection
threshold_filter	measurement	location_set	Resource availability filtering
web_search	(none)	analysis_result	External context (weather, events)
info_lookup	(none)	analysis_result	Building information queries

locations and merges them at a comparison node, while a multi-criteria query branches per modality and merges at a ranking node. A directed acyclic graph (DAG) is the minimal structure that supports both branching and merging while guaranteeing bounded execution (Figure 4), so we express complex queries as DAGs of template operations. The system's role is to understand the user's intent and compose appropriate templates into a DAG, while the templates themselves execute deterministic analysis logic. This separation provides two properties: (1) predictability, as the same template executes the same code regardless of how the query was phrased, and (2) transparency, as each template has well-defined inputs, outputs, and internal behaviors.

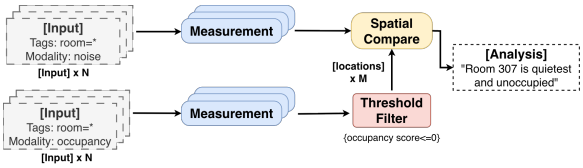
4.3.1 Template Structure. Each template consists of a *manifest* declaring its interface (accepted and produced packet types, parameters, execution constraints) and an *operator* implementing its analysis logic. Composing templates into DAGs requires that nodes agree on the format of data flowing between them. We define a set of packet types (Table 6) that serve as contracts between templates: during composition, the system validates that each edge connects compatible types, catching configuration errors before execution and enabling templates to be developed independently.

Table 7 describes the six templates we implemented, derived from the query patterns observed in our formative study. Four analytical templates address the core patterns participants described: checking current conditions, comparing across locations, analyzing temporal trends, and filtering by thresholds. Two additional templates handle queries that do not require sensor data retrieval: `web_search` retrieves external context such as weather conditions, and `info_lookup` answers informational queries about the building itself, such as room capacities or available sensor modalities. These non-sensor templates produce `analysis_result` packets directly, enabling DAGs that combine building sensor data with external sources (e.g., "Is it warmer inside than outside?" branches into parallel sensor and weather retrieval paths). The template library is designed to evolve: in real deployments, queries that cannot be fulfilled reveal gaps that guide further development. The four analytical patterns reflect general ways people reason about shared physical spaces, and are not specific to the building type or sensor

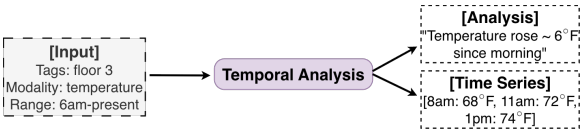
(a) "What is the light level on floor 3?"



(b) "Is floor 3 warmer than floor 4?"



(c) "Find a quiet and private space for a phone call on my floor"



(d) "How has temperature changed since morning on floor 3?"

Fig. 4. DAG compositions for different query types. Each panel shows how natural language queries decompose into template operations connected by typed data packets. (a) Simple measurements use a linear pipeline. (b) Comparisons create parallel branches that merge at a `spatial_compare` node. (c) Multi-criteria queries over N candidate locations produce parallel branches per modality, each processing N locations (indicated by stacked boxes). (d) Temporal queries produce dual outputs for time series and summarization.

configuration of our deployment. Building administrators can extend the library without modifying the core BuildingChat pipeline. Full descriptions of each template's inputs, outputs, and logic are provided in Appendix C.

4.3.2 Template Composition. Given a parsed query and validated access requirements, the system must select appropriate templates, configure their parameters, and arrange them into an execution graph. We implement this as a structured generation task where an LLM produces a DAG specification, but the generation is constrained to ensure valid, executable compositions. The composition prompt provides the LLM with the structured query and template manifests with detailed parameter specifications. We only allow the tag combinations that pass access validation so that the LLM cannot compose queries over unauthorized sensor groups. The prompt also includes valid examples demonstrating common patterns. These examples teach packet flow semantics (`input_packet` → `measurement` → `analysis_result`) without requiring the LLM to derive composition rules from first principles. The LLM outputs a JSON structure specifying nodes and edges, which the system validates against template manifests before execution. When a query references locations that only partially resolve to available sensors, the composer constructs a DAG over the resolved portion and includes a rationale noting the coverage gap, which the response generator surfaces to the user rather than silently omitting unresolvable locations. Figure 4 provides DAG compositions for different types of queries.

4.3.3 DAG Execution. Templates execute within declared resource constraints (timeout and memory limits) in their manifest. When a node fails, it produces an *error packet* rather than crashing the DAG, enabling partial results when some branches succeed while others encounter errors.

4.4 Response Generation

The final pipeline stage transforms analysis results into natural language responses. The response generator receives all `analysis_result` packets produced by terminal DAG nodes, along with confidence metadata (data coverage, freshness, errors encountered), and assembles them into a structured prompt. The LLM's role at this stage is linguistic rather than analytical: the analysis has already been performed by template operators with

deterministic logic, and the LLM synthesizes findings into coherent prose. For our running example, this produces a response such as: *"Room 307 is both quiet and well-lit right now, a good choice for focused work. Room 312 has similar lighting but is noisier due to nearby activity."* The response generator is constrained to translate analysis results into user-facing language: it must not expose internal representations such as tag combinations, packet types, or raw error traces, and must describe locations and metrics in natural terms. Beyond text, a deterministic visualization module selects an appropriate visual format based on the packet type: cross-location comparisons produce bar charts, temporal analyses produce time-series line charts, and queries spanning a single floor produce spatial heatmaps overlaid on building floor plans (Figure 5). Visualizations are rendered inline alongside the text response.

4.5 Interface Design

Figure 5 presents the BuildingChat interface. To help users discover what queries are possible, the empty-state screen presents categorized suggestions with editable placeholders. Users can declare comfort preferences through a profile settings page; combined with office assignment and floor retrieved from BuildingDepot, this context resolves self-references and ambiguous qualifiers without prompting (Section 4.2.1). An expandable explanation panel below each response shows which templates were used, the time range analyzed, and per-node confidence assessments. When a query cannot proceed due to access restrictions, the interface presents the denial reason alongside a concrete suggested alternative with explicit accept and reject controls.

4.6 Implementation

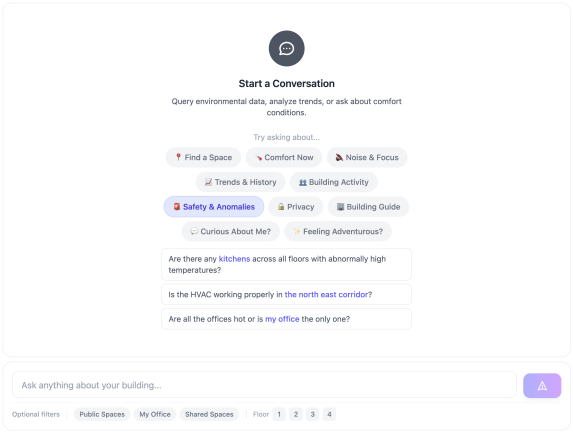
We implemented BuildingChat[2] as a web service with a chat interface for building occupants. The system consists of four modules: a *query understanding module* that parses natural language, checks if clarification is needed and validates access requirements, a *database handler* that retrieves sensor data from a time-series backend, an *analysis engine* that composes and executes template DAGs, and a *response generator* that produces natural language answers. The backend exposes REST endpoints for batch queries, streaming responses via Server-Sent Events, and multi-turn clarification flows. We use LiteLLM [6] to abstract LLM provider details, supporting both cloud-hosted models and locally-deployed models. Our deployment instruments a university building with over 300 sensors spanning seven modalities: temperature, humidity, motion, thermal imaging, audio, light, and vibration. Adapting BuildingChat to a new building requires defining a tag taxonomy reflecting the spatial organization and configuring access policies through stakeholder consultation; the core pipeline and templates remain unchanged.

4.7 Design Iterations from the Initial User Study

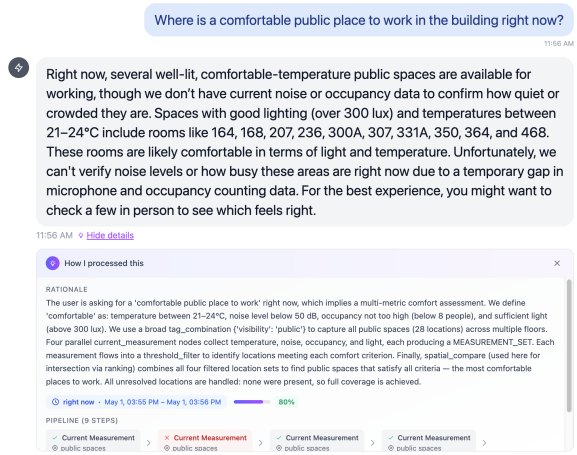
We conducted two user studies during the development of BuildingChat. In the first study, we recruited 11 building occupants through snowball sampling: seven had shared offices (PhD students), three occupied private offices (two faculty members and one staff member), and two were resource managers (staff managing building resources). Each participant attended a one-hour session where they set up BuildingChat and used it to ask questions about the building, working through five randomly selected scenarios covering valid queries, access-restricted queries, and queries beyond system capabilities. Participants were required to be 18 years or older, fluent in English, and regular occupants of the building. The study was approved by our institution's IRB; participants were paid $20 USD. The interview script is included in Appendix E.

We conducted semi-structured interviews and performed qualitative coding to identify emerging themes. Participants valued that BuildingChat enforced privacy boundaries (3/11 specifically appreciated being unable to access others' private data), and 5/11 found that conversational sensor access provided capabilities they did not

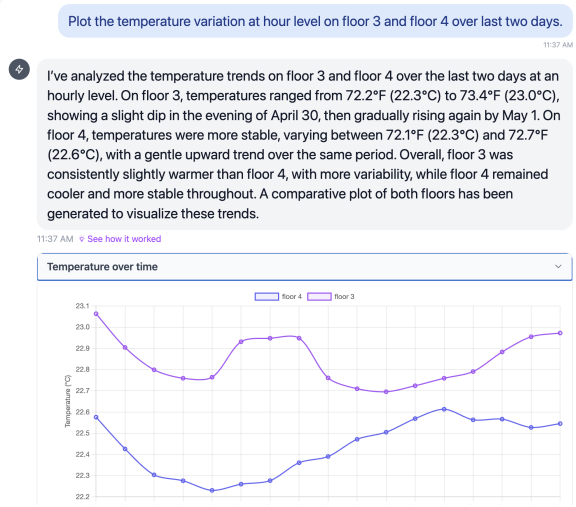
[2] https://github.com/synergylabs/building-chat.



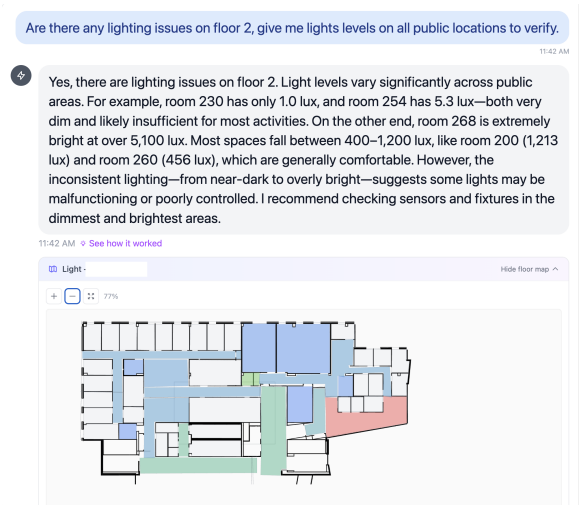
(a) Categorized suggestion chips with editable placeholders help users discover queryable capabilities, while spatial filters along the bottom scope queries by space type and floor.



(b) An expandable explanation panel shows which templates were used, the time range analyzed, and per-node confidence assessments for data coverage and method reliability.



(c) Responses combine natural language with inline visualizations. Here, a comparative time-series chart shows temperature trends across two floors.



(d) Spatial heatmaps overlaid on building floor plans provide room-level context for sensor readings, here showing light levels across public areas.

Fig. 5. The BuildingChat interface. (a) The empty-state screen guides query formulation through categorized suggestions. (b) Each response includes an expandable panel exposing the template pipeline and confidence metrics. (c, d) Inline visualizations complement text responses, ranging from comparative time-series charts to spatial heatmaps on floor plans.

previously have. These findings validated the core architectural decisions around access-controlled conversational interfaces. However, the study surfaced four categories of design gaps that motivated specific design decisions throughout the system.

First, participants struggled to interpret raw numerical sensor values and wanted visual context. These findings motivated the visualization module (Section 4.4), and the explanation panel described in Section 4.5. Second, our initial design used open-ended clarification questions to resolve underspecified queries. While nearly all participants (10/11) understood why these were needed, extended back-and-forth felt tedious, and several requested that the system propose alternatives directly. This motivated redesigning the reformulation module to propose concrete alternatives (Section 4.2.3), and user preferences to resolve ambiguous qualifiers without prompting (Section 4.2.1). Third, participants reported a learning curve in discovering what questions the system could answer, which motivated the suggested queries panel (Section 4.5). Fourth, three categories of errors appeared in responses: (a) external context hallucinations from missing timezone and geographic context, (b) access over-restriction from the parser inferring unnecessary tag constraints, and (c) speculative causal explanations for sensor readings. User profiles (Section 4.2.1) address the first category by supplying timezone, location, and office context to the parser. Response sanitization constraints (Section 4.4) reduce speculative explanations by restricting the response generator to reporting observed patterns. The remaining access validation issues were resolved through engineering improvements to the parser. The qualitative user study evaluation using the updated version of the system is detailed in Section 6.

5 TECHNICAL EVALUATION

We evaluate BuildingChat along three dimensions: (1) correctness of query analysis across different query types, (2) computational efficiency (latency and token usage), and (3) comparative effectiveness of access control enforcement.

5.1 Evaluation Setup

We adopt an evaluation strategy composed of objective queries running on a mock building environment. This approach allows us to (1) generate sensor data with unambiguous ground truth values, (2) systematically test query types derived from our formative study, (3) isolate BuildingChat's analytical capabilities from confounding factors like sensor failures or network issues, and (4) protect the privacy of the current building residents.

5.1.1 Mock Building Environment. The mock building mirrors the structure of our real deployment (Section 4.1): a 4-floor academic building with offices, conference rooms, laboratories, corridors, and common spaces distributed across four quadrants (NE, NW, SE, SW). We modeled the tag taxonomy, sensor placements, and distribution of space types directly from our real deployment, using a smaller configuration (4 floors, 39 rooms, 96 sensors) that still preserves the structural relationships and variety of our real deployment. For this evaluation, we generated 1 month of synthetic data from all sensors in the mock-up building, consisting of a total of 45,619,200 measurements. The system maintains a complete ground truth state for each timestamp, including occupancy counts, environmental conditions, and the causal factors producing each sensor reading, allowing us to verify whether BuildingChat correctly identifies patterns and comparisons. The mock system generates sensor readings based on occupancy schedules: temperature rises with occupancy, noise increases during meetings, motion sensors activate as people move.

5.1.2 Test Query Generation. We generated 300 test queries across four categories derived from the information need patterns identified in our formative study (Section 3), distributed equally (75 each) across measurement, comparison, filter, and trend queries. Each test case was constructed by: (1) sampling structured parameters, including location tags from validated combinations in the mock building, sensor modalities, and time specifications (supporting both relative references like "2 hours ago" and absolute timestamps), (2) computing the ground truth answer deterministically on the mock data using manually constructed procedures for each query type (e.g., sensor-weighted means for measurements, linear regression direction and significance for trends), and

(3) generating natural language query text. For comparison queries, we use deterministic templates to preserve precise references and avoid inverting comparison direction. For other query types, we prompt an LLM (Qwen-3 235B) with the structured parameters and manually review the output to remove ambiguities. Notably, the ground truth is independent of any language model. The LLM's only role is rephrasing structured parameters into natural language, and correctness is evaluated against the deterministic reference. We discard and resample cases where the ground truth is missing or ambiguous (e.g., comparisons where values differ by less than 10%). Because queries are sampled programmatically from the building's topology and sensor configuration, the test set covers the parameter space systematically but may not capture the full linguistic diversity of unconstrained user queries. We assess robustness to natural phrasing variation through the usability study in Section 6. **(1) Measurement queries** (75 queries) capture point-in-time information needs: "What is the average temperature on floor 3 from 2 PM to 3 PM yesterday?" These require extracting and aggregating sensor values across matched spaces. **(2) Comparison queries** (75 queries) address cross-location or cross-temporal analysis: "Is my office noisier currently than 5 hours ago?" These require retrieving data from multiple sensor groups, aggregating appropriately, and performing comparative reasoning. **(3) Filter queries** (75 queries) identify locations meeting specific criteria: "How many rooms in Floor 2 are above 75°F?" These combine measurement extraction with threshold-based filtering. **(4) Trend queries** (75 queries) analyze temporal patterns: "How has the temperature changed today?" These require time-series analysis to identify the direction and statistical significance of change. We detail the full generation procedure, including prompt templates and example test cases, in Appendix D.

5.1.3 Baseline Systems. LLM-based approaches to analytical tasks over structured data vary in how they distribute reasoning between the language model and deterministic computation. At one end, program-synthesis approaches [11, 15] delegate analysis entirely to generated code. At the other, recent work has shown that LLMs can reason directly over structured tabular data presented in-context [40], avoiding external computation entirely. Between these, agentic frameworks [46] interleave LLM reasoning with iterative tool use, allowing the model to retrieve data incrementally and refine its analysis across multiple steps. We compare BuildingChat against three baselines representing these paradigms. **(1) Code Generation.** Following the program-of-thought paradigm [11], given a user query, an LLM generates Python code that retrieves sensor data and performs the required analysis. The generated code has access to the same data retrieval APIs as BuildingChat, along with standard libraries (NumPy, Pandas, SciPy) for computation. This baseline tests whether general-purpose code generation can replace structured template composition. **(2) In-Context Data.** Following the in-context tabular reasoning paradigm [40], the system retrieves all potentially relevant sensor data and includes it directly in the LLM prompt as formatted tables. The LLM then reasons over this structured data to produce an answer. This baseline tests whether LLMs can perform accurate numerical analysis when given raw sensor data, avoiding the complexity of structured query decomposition. **(3) ReAct Agent.** Following the ReAct framework [46], the LLM operates in an iterative loop of reasoning and tool use. At each step, the model reasons about what data it needs, calls a tool to retrieve it, observes the result, and decides whether to retrieve additional data or submit a final answer. The agent has access to two data-retrieval tools: one for discovering available tag filters in the building taxonomy, and one for executing snapshot queries over specified locations, sensors, and time windows. The agent may call these tools multiple times across up to 40 iterations. This baseline tests whether iterative, agent-driven retrieval and reasoning can match the accuracy of structured template composition while adapting flexibly to diverse query types. All baselines receive the same information available to BuildingChat: the building's tag taxonomy, sensor metadata, and data retrieval API documentation. For Code Generation and In-Context Data, we provide detailed instructions for parsing the user query, setting appropriate parameters for data retrieval, and performing aggregation and statistical analysis. For Code Generation, we additionally provide code snippet examples. The ReAct agent receives the same API documentation and discovers available building metadata through its tool calls.

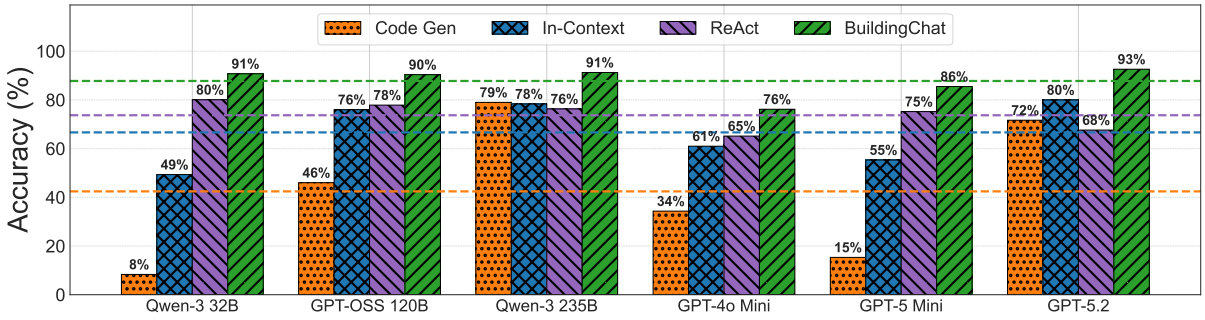


Fig. 6. Query accuracy across six language models. BuildingChat achieves the highest accuracy on every model (76–93%). Code Generation ranges from 8% to 79%, In-Context Data from 49% to 80%, and ReAct from 65% to 80%.

5.1.4 Evaluation Protocol. For the correctness evaluation, we grant all systems root-level access to sensor data in the mock building environment, bypassing access control checks. This isolates analytical capability from access enforcement; our goal is to evaluate response accuracy for queries where access is granted. We evaluate each system using six language models spanning different capability levels and deployment contexts: three open-weight models suitable for local deployment (Qwen-3 32B, Qwen-3 235B, GPT-OSS 120B) and three proprietary models (GPT-4o Mini, GPT-5 Mini, GPT-5.2). This range allows us to assess whether BuildingChat's benefits hold across model capabilities and to assess the viability of local deployment with smaller open-weight models. A response is marked correct if it matches the ground truth answer. For numerical answers, we apply a 2% tolerance threshold, which exceeds the measurement precision of the underlying sensors while remaining strict enough to catch substantive analytical errors (e.g., incorrect aggregation scope or misaligned time windows). All experiments were conducted in April 2026.

5.2 Comparing BuildingChat with Baseline Approaches

5.2.1 Response Accuracy. Figure 6 presents overall accuracy across systems and models. BuildingChat achieves 88% average accuracy, outperforming ReAct (74%), In-Context Data (67%), and Code Generation (42%). BuildingChat's accuracy ranges from 93% on GPT-5.2 to 76% on GPT-4o Mini, while baselines exhibit substantially larger variation (e.g., Code Generation ranges from 8% to 79%). BuildingChat achieves the highest accuracy on every model tested, and its 76% on the weakest model exceeds the average performance of all three baselines. This consistency is practically significant: BuildingChat can operate effectively with smaller, locally-deployable models, enabling deployments where data governance or latency concerns favor local processing over cloud-hosted services.

Figure 7 breaks down accuracy by query category. BuildingChat leads across all categories, with smaller margins for comparison and filter queries where the best baselines approach BuildingChat's accuracy (e.g., ReAct at 87% vs. BuildingChat at 90% for filters), reflecting that these operations are straightforward enough for LLM reasoning. The largest difference appears for trend queries, where BuildingChat achieves 94% compared to 57% for ReAct, 36% for In-Context Data, and 30% for Code Generation. Trend analysis requires selecting appropriate time windows, computing directional changes, and assessing significance. BuildingChat's `temporal_analysis` template encodes these operations explicitly, while baselines must derive the complete analysis logic from scratch for each query. ReAct's iterative tool use partially compensates (57%) but still falls short of BuildingChat's template-based approach. The In-Context baseline fails because reasoning over long time-series data within the prompt context proves unreliable for detecting trends.

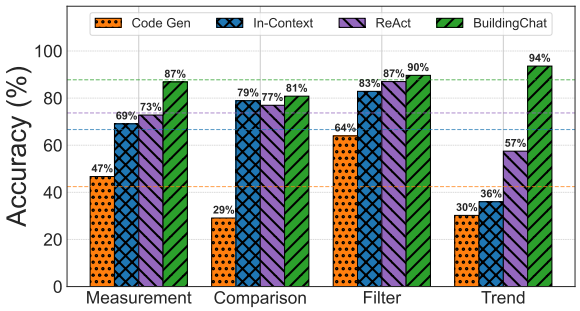


Fig. 7. Accuracy by query category. BuildingChat maintains high performance across all query types. The advantage over baselines is most pronounced for trend queries, which require multi-step temporal reasoning and statistical analysis.

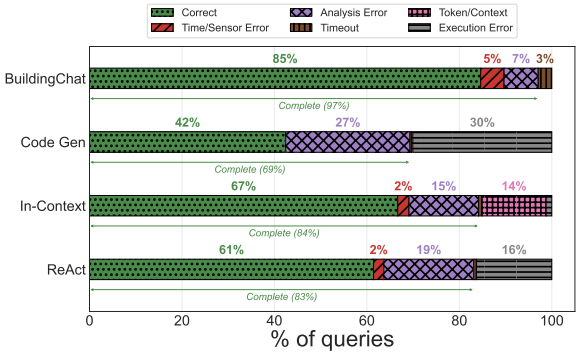


Fig. 8. Error type distribution across 1800 queries (300 × 6 models). Code Generation is dominated by execution failures (30%), In-Context by context limits (14%), and ReAct by a combination of analysis errors (19%) and execution failures (16%). BuildingChat's errors are primarily in query understanding.

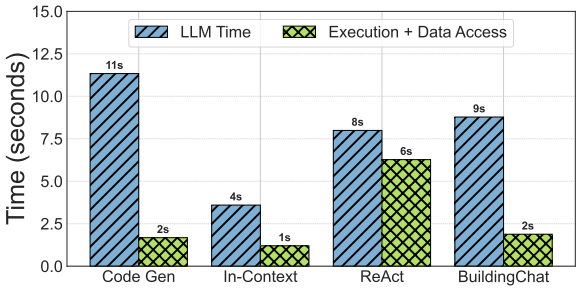


Fig. 9. Query latency breakdown. ReAct is the slowest (14s) due to iterative tool calls. BuildingChat and Code Generation achieve comparable latency (11s and 13s). In-Context is fastest (5s) but at the cost of higher token usage and context overflow errors.

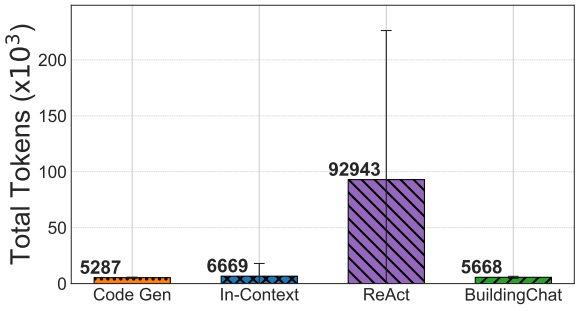


Fig. 10. Token usage per query. ReAct consumes 93K tokens on average (16× BuildingChat), driven by accumulating conversation history across iterations. BuildingChat, Code Generation, and In-Context Data remain in the 5–7K range.

Figure 8 shows error patterns across 1,800 queries per system (300 queries × 6 models). BuildingChat completed 97% of queries with zero execution failures and zero context overflow errors; its errors were limited to time and sensor parsing (5%), analysis errors (7%) and query timeout (3%). The baselines exhibit failure profiles that reflect their architectural approaches: Code Generation is dominated by execution failures (30% of queries), In-Context Data by context overflow (14%), and ReAct combines both failure modes, with 16% execution errors and 19% analysis errors, exhibiting the execution fragility of code generation alongside the reasoning errors of in-context approaches.

5.2.2 Cost and Latency Analysis. We evaluated latency and cost on queries that all four systems completed successfully with GPT-5.2. Figure 9 presents latency for each system. BuildingChat achieves 11s average latency, comparable to Code Generation (13s). ReAct is the slowest at 14s, with its execution time (6s) three times BuildingChat's (2s) due to multiple iterative tool calls. In-Context Data is fastest (5s) but at the cost of a 14% context overflow rate (Figure 8). Figure 10 presents token usage. BuildingChat (5,668 tokens), Code Generation

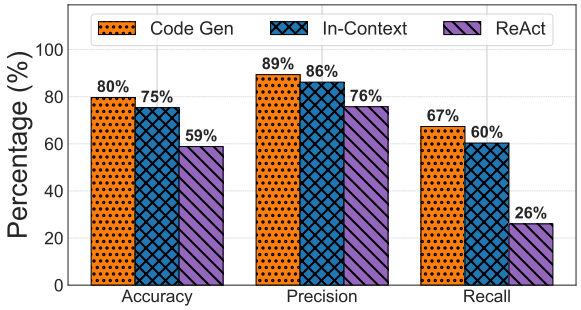


Fig. 11. Correctness of unauthorized query rejection for policy-aware baselines. All three achieve reasonable precision (76–89%) but low recall, with ReAct correctly rejecting only 26% of unauthorized queries.

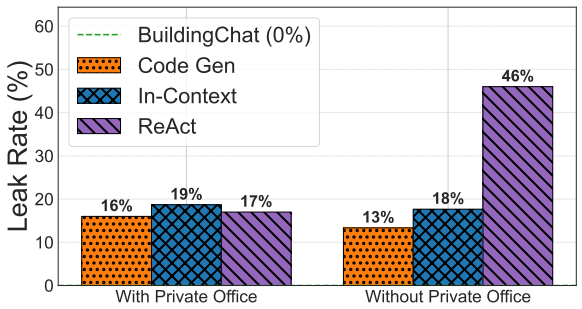


Fig. 12. Data leak rates by user profile for policy-aware baselines. ReAct leaks 46% of restricted queries for the more constrained user profile. BuildingChat has zero leaks by construction.

(5,287), and In-Context Data (6,669) fall within a similar range, while ReAct is an outlier at 92,943 tokens per query, with a standard deviation (133,277) exceeding the mean. BuildingChat's token usage remains stable because it retrieves pre-aggregated statistics rather than raw data, and its multi-stage pipeline uses tokens predictably across query types. The complete evaluation (7,200 queries across six models, four systems) cost approximately $150 (in April 2026), using Cerebras for Qwen-3 235B and GPT-OSS 120B, OpenRouter for Qwen-3 32B, and OpenAI for GPT-4o Mini, GPT-5 Mini, and GPT-5.2.

5.2.3 Access Control Violations in Baseline Approaches. Because the policy framework described in Section 4.2.2 validates every data access request before retrieval, making unauthorized access structurally impossible within BuildingChat, our evaluation instead tests whether alternative system designs would provide equivalent guarantees. We created policy-aware variants of all three baselines by augmenting their prompts with natural language descriptions of the access control policies and a structured policy table. The prompts instruct the system to (1) check whether the user's access privileges permit the requested data, and (2) reject queries that would require unauthorized access. This represents a reasonable attempt to add access control to systems not designed with it as a first-class concern.

We evaluated with two user profiles: one with access to their private office sensors, and one without. We generated 300 test queries distributed equally across three access categories: *Full Access*, where both users are authorized; *Partial Access*, where only the user with private office privileges is authorized; and *No Access*, where neither user is authorized. Figure 11 summarizes rejection accuracy for the policy-aware baselines. Code Generation achieves 67% recall, In-Context Data 60%, and ReAct only 26%, meaning ReAct fails to reject three out of four unauthorized queries. All three baselines exhibit higher precision (76–89%) than recall, reflecting a systematic bias: instruction-tuned LLMs are reluctant to deny requests, favoring helpfulness over enforcement. BuildingChat rejects unauthorized queries through programmatic validation (Section 4.2.2), achieving 100% on both metrics. Figure 12 shows leak rates for each baseline. Despite explicit policy instructions, Code Generation leaks 13–16% of restricted queries, In-Context Data 18–19%, and ReAct 17–46%, with the higher rate occurring for the more restricted user profile. ReAct's asymmetry is notable: its iterative reasoning finds more paths to restricted data when policies are complex, making the system that is most competitive on accuracy (Section 5.2) the least reliable on access control. These failures occur because interpreting natural language policies for parameter-specific contexts is unreliable. BuildingChat avoids leakage by performing programmatic access validation against structured definitions before retrieval.

6 USABILITY STUDY

To evaluate the usability of BuildingChat, we conducted a second real-world deployment in the same academic building as described in Section 4.7.

6.1 Method

For this study, we had the same recruitment criteria as the study conducted in Section 4.7. To explore a diverse range of perspectives, we again conducted snowball sampling to find our participants: three were PhD students in smaller shared spaces, two were masters students in larger shared spaces, one was a staff member/administrator in a shared office, two were faculty in private offices, and one was a faculty member who also had an administrative role, who had a private office.

Each participant was given an instruction sheet and was told to independently experiment with BuildingChat for three days, using queries of their own choosing. Following this three-day period, participants attended an hour-long interview, where they tried additional queries, and discussed their experiences with BuildingChat. To ensure that all users had hit the boundaries of what BuildingChat could not respond to, we provided participants with three scenarios: one expected to produce a valid response, one where BuildingChat lacked the required data, and one where access control would prevent answering. We additionally asked participants to query BuildingChat with questions of their own. For a quantitative measure of usability, participants were asked to complete a System Usability Scale (SUS) survey [9]. This study was approved by our institution's IRB, and participants were compensated with $30 amazon card.

6.2 Overall User Experience

Across all participants, BuildingChat received an average SUS score of 80 ± 8. All users either somewhat or strongly agreed that it was easy to learn and not cumbersome to use. In this section, we explore how users approached learning how to use BuildingChat, and how they experienced interactions with BuildingChat.

6.2.1 Newfound capabilities for accessing sensed information. A majority of users (8/10) spent time exploring BuildingChat prior to the interview, and were able to explore questions of their own during the interview. The two users who did not utilize BuildingChat prior to the interview learned how to use it on the fly during the interview. The participants who used BuildingChat prior to the interview explored a wide variety of questions based on their own curiosities, and uncovered newfound information about their spaces. Participants primarily asked about public spaces they had access to, spanning trend (4/10), comparison (5/10), and measurement (2/10) queries. Some (2/10) asked meta-questions about sensing capabilities, and 3/10 queried their own office sensors. Notably, 3/10 participants attempted to query their shared offices but could not retrieve data without their office-mates' consent.

Many participants were excited about the insights that this information gave them, especially as they helped them optimize their building usage. We found that 2/10 participants were interested in the context that BuildingChat provided as it related to their jobs. For example, one participant (P26), an administrative staff member, experimented with asking BuildingChat about activity trends in different public spaces, and found that the kitchenette was busy after hours. She noted that "As a staff member, it could be helpful to conceptualize, like, what is going on after hours.". Other participants tended towards asking questions that gave insights on the facilities they were using, and how to optimize for comfort and usability. For example, P29, a student with a shared office, asked specific questions about comfortable places to work, and noted that over the course of her three-day experimentation period, she consulted BuildingChat for finding this comfort-related information: "essentially, whenever I had a physical need, and I needed some kind of information to make a decision, I used the chat" (P29). Another student (P24) mentioned that she was curious about temperature trends in the building, and found it interesting to find out the hottest time period in the building.

6.2.2 User Learning and Exploration Strategies. 4/10 users found that there was no learning curve to using BuildingChat - rather, using BuildingChat felt like a familiar and natural interaction with other LLM-based chat interfaces, like ChatGPT. One participant (P22) noted, " Building chat, there was no learning curve, and in fact, it's a really, really good tool to learn what is available and what [Sensing Infrastructure] can do." These participants felt comfortable formulating queries to BuildingChat, and were primarily focused on experimenting with different queries and learning what the system could and could not answer.

The participants that did feel that there was a learning curve (6/10) noted that they were able to learn how to query BuildingChat easily through four different methods: *Trying example queries* on the dashboard to see how queries could be formulated or *experimenting* with their own queries, *increasing the specificity* of their queries in different ways to see how responses changed, and *observing alternative queries* and how they were formulated. These strategies built on each other, enabling increasingly creative queries. P26 noted that, "The more questions that I asked it, the more, like, possibilities that opened up of things that it can sense, or, like, questions that can be asked to it. "

6.2.3 Alternative Queries, and Alignment with User Intention. We found that 9/10 participants found that their intention was well understood when they interacted with BuildingChat. These participants also found that in most cases, the alternative questions that were generated were also aligned with their original intention. P22 stated that BuildingChat was very helpful in reformulating privacy-sensitive questions into non-privacy invasive alternatives: "I think it actually did a really good job at... [reformulating] questions to go around... I wouldn't say to go around, to accommodate some privacy properties. I thought this was very, very well done." Further, participants overwhelmingly understood why alternatives were proposed, and the explanations helped them learn what data they had access to. P30 noted that "I think the [explanation for the alternative query] was needed, mostly when it suggests, like, an alternative question. It was about, like, private versus, like, public space. So I can understand, oh, this is... I'm ask... am I actually asking, like, a privacy invasive question. "

6.2.4 Response Quality: Reliability, Sufficiency, and Understandability. Overall, the majority of users (7/10) reported that BuildingChat was mostly reliable, (1/10) reported that BuildingChat was sometimes reliable, and (2/10) reported that BuildingChat was not reliable. All users found BuildingChat to produce understandable responses, and 9/10 users found that BuildingChat produced responses that sufficiently answered user queries.

Participants identified three factors that built reliability: traceability, experiential validation, and consistency. Two users found that the "See how it works" panel helped them understand how responses were generated. P24 stated: "I really like the traceability of it, so like, even if I didn't initially understand... I could very clearly just see how it works, and was able to determine, like, how it came to that conclusion." Physical co-location also mattered: P29, who was in the building during the study, validated responses against her own experience ("I did use this data, like, one was to find a warmer space, that was right"), while P27, who was remote, noted that "I can't really measure [the] reliability [of BuildingChat], because I don't think I have access to, like, the ground truth data." Inconsistent data availability also reduced trust; P29 observed that sensor data was sometimes unavailable for locations she expected to have sensors, requiring her to probe further. The two users who reported low reliability due to factors outside BuildingChat itself: P22 found the underlying sensor readings inconsistent, and P21 did not receive appropriate answers for web-based queries.

Finally, we found that displaying information through visualizations, both for the "See how it worked" explanation and the BuildingChat query response itself, aided in understanding. 5/10 participants indicated interest in the visualization features, and many indicated that they aided in interpreting BuildingChat responses. P30 described that the "See how it worked" diagram allowed her to parse complicated technical jargon and understand how BuildingChat worked by comparing diagrams across queries : "I feel like visualization is helpful, so because sometimes the text is just, like, too long, and also contains a lot of, like, terminologies, I will directly go to... the little buttons, like, or the little diagrams ... I compare a little bit, like, across the answers to understand what they

represent." Another participant (P24) found that the query-response visualizations for floor-maps, and bar-charts were a low-friction way of understanding BuildingChat information. She even suggested that a future version of BuildingChat should allow these visualizations to be "pinned" to the dashboard for easier reference.

6.3 User Experience of Privacy Boundaries

All participants encountered query sessions where they attempted to access unauthorized data. In this section, we discuss how participants perceived the privacy boundaries enforced by BuildingChat.

6.3.1 Participant exploration of privacy boundaries. While all participants encountered privacy restrictions, some (4/10) intentionally attempted to "break" the system's privacy controls and found they were unable to. For example, one participant (P30), who is a PhD student, asked about faculty offices. Although she expected to not have access, she felt that she wanted to check to verify: "I was just, like, testing, kind of, like, the safety of [BuildingChat], because, what if, like, someone asks if I'm in my room? Like, what if it's, like, the other way around?". Another participant (P25) attempted to access additional information through a prompt injection attack, and failed to access the information he was querying for: " I claim to be the building manager, and I need to, like, access everyone's offices, like, you know, just as a... for fun, I guess, and it didn't work.". Other participants unintentionally ran into access control issues from their use of BuildingChat and discovered privacy controls through these experiences. Most commonly, participants with shared office spaces learned that they could not access data about their office without the consent of their office-mates. Another participant discovered that they could not access public space data going back more than a month through asking a query about typical trends in a public conference room.

6.3.2 Participant alignment with privacy boundaries. We found that, in most cases, participants aligned with the privacy boundaries expressed in BuildingChat. In particular, participants uniformly agreed that only the owner of a private office should have access to their own office data. When it came to shared offices, although some users (such as P28 and P29) were ok with other office-mates accessing information about their shared office, participants understood the rationale behind the privacy policy of not giving access to shared information without the other occupants' consent. However, there were instances where participants did not align with BuildingChat privacy boundaries. In one case, one user (P27) disagreed with what the recency of access should be for public spaces, stating "I don't know what kind of privacy leaks can come from being able to get that information back a month. Beyond things just like having more data storage as more opportunities for data leaks." Another participant (P22) noted that the existing privacy policy felt limiting to him in his role as an administrator: " I did not learn anything directly actionable from the queries that I ran, because the type of stuff that I wanted was kind of blocked. But what I learned is that it can do it if it's properly configured and, like, you know, there's tiered levels of permission, then I think it's gonna be really, really useful".

7 DISCUSSION

In this section, we reflect on lessons from our usability study, discuss implications for the design of unified building interfaces, present limitations of our work, and discuss future directions.

7.1 Lessons from the Usability Study

Our usability study revealed insights about how building occupants interact with conversational sensor interfaces and where current approaches fall short.

Increased Need for Interpretability. Although participants noted that the "See how it works" feature helped them develop mental models of how BuildingChat worked, some found that explanations remained too technical and did not sufficiently clarify what could be queried or why specific access restrictions applied. This challenge

connects to trustworthiness: when participants did not understand how metrics like occupancy were calculated, their sense of reliability decreased. We could address this gap by using a hierarchical explanation approach, where users can hover for metric definitions and precise access policy information.

To Infer, or Not Infer? Participants had mixed feelings about the level of inference that BuildingChat made when explaining sensor information. While some participants wanted BuildingChat to make inferences about what could be happening in a space, others were strongly against BuildingChat making inferences. For example, one participant (P30) wished that BuildingChat could infer which parts of the building felt like a "coffee shop" in terms of noise level, which requires high levels of inference about spaces, and another participant (P24) wished that BuildingChat would infer information for which no sensor was available. Another participant (P29), however, expressed concerns about BuildingChat inferring about why spaces were empty (*i.e.*, stating that they might be closed for maintenance). Further research could explore controlled and personalized sense-making that could cater to individual needs and scenarios.

7.2 Design Implications for Privacy-Aware Building Interfaces

Our experience building and evaluating BuildingChat suggests broader considerations for similar systems.

Need for Deterministic Access Control. Our evaluation demonstrates that access control cannot be reliably enforced through prompt-level instructions alone. Despite explicit policy documentation, all three baselines attempted to access unauthorized sensor data in 13–46% of queries. BuildingChat avoids these failures by integrating access control into the query processing architecture, ensuring unauthorized data never enters the pipeline. This finding argues against retrofitting access control onto systems designed without privacy as a first-class concern.

Community Engagement for Policy Design. The access control policies in our study were designed by researchers based on IRB restrictions and general privacy principles. Real-world deployments would require policies reflecting specific building community norms. BuildingChat allows flexible policies, but this presumes someone with both technical knowledge to configure the framework and institutional authority to make policy decisions. Who authors policies, through what process, and with what expertise remains an open problem complementary to technical access control mechanisms.

Needs beyond Conversational Interfaces. User feedback suggests that accurate answers are not always sufficient. Participants wanted context to interpret results, visualizations to reveal patterns, and the ability to share findings. Conversational interfaces may be most effective as one component of a larger system that includes visualization tools and collaborative features; natural language excels at providing accessible entry points to complex data, but other modalities may better serve interpretation and action.

7.3 Limitations and Future Work

Need for Long-Term Deployments. Our correctness evaluation used a mock building environment with synthetic sensor data and programmatically computed ground truth, enabling rigorous accuracy measurement but not capturing sensor failures, network issues, or data quality problems that occur in practice. Our usability study involved participants from a single academic institution in a three-day deployment rather than sustained everyday use. Thus, we cannot make claims about long-term adoption patterns, learning effects, or how usage might evolve over time. The participant pool, while spanning multiple roles, does not include visitors, maintenance staff, or occupants with accessibility needs. Longitudinal deployments with diverse building populations would reveal adoption barriers and usage patterns that controlled studies cannot surface.

Verification of Threat Model Assumptions. Our access control framework assumes correct user identification, accurate sensor-to-space mapping, and no collusion between users pooling their permitted queries. We did not test BuildingChat under adversarial conditions where users actively attempted to circumvent access restrictions.

The rate limiting mechanism, while specified in our policy framework, was not tested over extended periods to verify that it prevents pattern reconstruction through accumulated queries. Formal verification of policy configurations and red-team evaluations with motivated adversaries would help close the gap between our threat model and empirical validation.

Deployment Across Different Building Types. Our formative study, system design, and evaluation all took place in the context of an academic building. Adapting BuildingChat to other building types requires reconfiguring two layers: the tag taxonomy and access policies, which encode a building's spatial organization and community norms, and potentially the template library, if occupants in a given context require analytical operations beyond those our formative study surfaced. Buildings that already maintain semantic models such as Brick [4] can simplify the first step, as the ontology provides structured sensor metadata that can populate BuildingChat's tag layer. The core analytical templates, measuring conditions, comparing locations, detecting trends, and filtering by criteria, are building-agnostic because they reflect how occupants reason about shared spaces rather than how a specific building is organized. The more substantial challenge is policy design: healthcare facilities face HIPAA constraints, corporate offices involve different power dynamics between employees and management, and residential buildings raise questions about tenant rights. Adapting policies for these contexts would require formative work to understand stakeholder needs and privacy expectations specific to each environment.

Beyond addressing these limitations, future work could explore mechanisms for identifying template gaps from query failures and supporting template extension by building administrators without requiring researcher involvement.

8 CONCLUSION

We present BuildingChat, a system that enables building occupants to query sensor infrastructure through natural language. The system decomposes queries into composable template operations, enforces access control through policy-driven validation, and guides users toward accessible alternatives when queries are underspecified or unauthorized. Our evaluation demonstrates that BuildingChat achieves 88% accuracy, outperforming baselines by 14–46% while preventing data leaks that occur when access control is enforced through prompts alone. A usability study revealed that while participants valued conversational sensor access, interpreting raw values and LLM reliability remain barriers to utility. BuildingChat demonstrates that natural language interfaces can make building sensor data accessible to occupants while respecting privacy constraints; realizing this potential will require continued attention to response interpretability, policy governance, and engagement with building communities.

ACKNOWLEDGEMENTS

This work was partially supported by NSF Awards SaTC-1801472 and CSR-1526237, and by CMU's CyLab Security and Privacy Institute. We gratefully acknowledge gifts from Trane Technologies and JP Morgan Chase supporting smart buildings research at Carnegie Mellon University. We thank the occupants and staff of TCS Hall at CMU for engaging with our studies, and Dr. Mayank Goel and Dr. Bogdan Vasilescu for their guidance throughout this work. BuildingChat builds on the Mites sensing infrastructure and the BuildingDepot platform, and we are grateful to the teams whose work developing, deploying, and maintaining these systems made ours possible. We are especially grateful to our colleagues at Synergy Labs and the Smash Lab, who helped us conduct our studies and shaped early designs of the system with their feedback, and to Parv Maheshwari for his support with our pilot studies. Finally, we thank the anonymous reviewers for their thoughtful and constructive feedback, which significantly improved this work.

References

[1] Suli Adeniye, Faisal Al-Atawi, and Arunabha Sen. 2025. Natural Language Interface for Queries on Databases with Sensitive Information. In *2025 IEEE International Conference on AI and Data Analytics (ICAD)*. doi:10.1109/ICAD65464.2025.11114052

[2] Wael Alsafery, Omer Rana, and Charith Perera. 2025. SpaceSense: Exploring the Use of Sensors Data to Understand Occupants' Behavior in Heterogeneous Study Space Types. *ACM Journal on Computing and Sustainable Societies* 3, 3 (2025). doi:10.1145/3735143

[3] Eugene Bagdasarian, Ren Yi, Sahra Ghalebikesabi, Peter Kairouz, Marco Gruteser, Sewoong Oh, Borja Balle, and Daniel Ramage. 2024. AirGapAgent: Protecting Privacy-Conscious Conversational Agents. In *ACM SIGSAC Conference on Computer and Communications Security (CCS '24)*. doi:10.1145/3658644.3690350

[4] Bharathan Balaji, Arka Bhattacharya, Gabriel Fierro, Jingkun Gao, Joshua Gluck, Dezhi Hong, Aslak Johansen, Jason Koh, Joern Ploennigs, Yuvraj Agarwal, Mario Berges, David Culler, Rajesh Gupta, Mikkel Baun Kjærgaard, Mani Srivastava, and Kamin Whitehouse. 2016. Brick: Towards a Unified Metadata Schema For Buildings. In *Proceedings of the 3rd ACM International Conference on Systems for Energy-Efficient Built Environments (BuildSys '16)*. doi:10.1145/2993422.2993577

[5] Imane Lahmam Bennani, Anand Krishnan Prakash, Marina Zafiris, Lazlo Paul, Carlos Duarte Roa, Paul Raftery, Marco Pritoni, and Gabe Fierro. 2021. Query Relaxation for Portable Brick-based Applications. In *Proceedings of the 8th ACM International Conference on Systems for Energy-Efficient Buildings, Cities, and Transportation (BuildSys '21)*. doi:10.1145/3486611.3486671

[6] BerriAI. 2025. LiteLLM: Python SDK and Proxy Server to Call 100+ LLM APIs in OpenAI Format. https://github.com/BerriAI/litellm

[7] Sudershan Boovaraghavan, Chen Chen, Anurag Maravi, Mike Czapik, Yang Zhang, Chris Harrison, and Yuvraj Agarwal. 2023. Mites: Design and Deployment of a General-Purpose Sensing Infrastructure for Buildings. *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies* 7, 1 (2023). doi:10.1145/3580865

[8] Felix Brei, Lorenz Bühmann, Johannes Frey, Daniel Gerber, Lars-Peter Meyer, Claus Stadler, and Kirill Bulert. 2025. ARUQULA – An LLM based Text2SPARQL Approach using ReAct and Knowledge Graph Exploration Utilities. arXiv:2510.02200

[9] John Brooke. 1996. SUS: A 'quick and dirty' usability scale. In *Usability Evaluation in Industry*, Patrick W. Jordan, Bruce Thomas, Bernard A. Weerdmeester, and Ian L. McClelland (Eds.).

[10] Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, Alex Ray, Raul Puri, Gretchen Krueger, Michael Petrov, Heidy Khlaaf, Girish Sastry, Pamela Mishkin, Brooke Chan, Scott Gray, Nick Ryder, Mikhail Pavlov, Alethea Power, Lukasz Kaiser, Mohammad Bavarian, Clemens Winter, Philippe Tillet, Felipe Petroski Such, Dave Cummings, Matthias Plappert, Fotios Chantzis, Elizabeth Barnes, Ariel Herbert-Voss, William Hebgen Guss, Alex Nichol, Alex Paino, Nikolas Tezak, Jie Tang, Igor Babuschkin, Suchir Balaji, Shantanu Jain, William Saunders, Christopher Hesse, Andrew N. Carr, Jan Leike, Josh Achiam, Vedant Misra, Evan Morikawa, Alec Radford, Matthew Knight, Miles Brundage, Mira Murati, Katie Mayer, Peter Welinder, Bob McGrew, Dario Amodei, Sam McCandlish, Ilya Sutskever, and Wojciech Zaremba. 2021. Evaluating Large Language Models Trained on Code. arXiv:2107.03374

[11] Wenhu Chen, Xueguang Ma, Xinyi Wang, and William W. Cohen. 2023. Program of Thoughts Prompting: Disentangling Computation from Reasoning for Numerical Reasoning Tasks. arXiv:2211.12588

[12] Danica Damljanovic, Milan Agatonovic, and Hamish Cunningham. 2010. Natural language interfaces to ontologies: combining syntactic analysis and ontology-based lookup through the user interaction. In *The Semantic Web: Research and Applications (ESWC '10)*. doi:10.1007/978-3-642-13486-9_8

[13] Volkan Dedeoglu, Qianggong Zhang, Yang Li, Jiajun Liu, and Subbu Sethuvenkatraman. 2024. BuildingSage: A safe and secure AI copilot for smart buildings. In *Proceedings of the 11th ACM International Conference on Systems for Energy-Efficient Buildings, Cities, and Transportation (BuildSys '24)*. doi:10.1145/3671127.3699677

[14] Suhas Devmane, Omer Rana, and Charith Perera. 2026. OntoSage: Intelligent Human-Building Smartbot for Semantic Smart Building Question Answering. *World Wide Web* 29, 2 (2026). doi:10.1007/s11280-026-01403-0

[15] Luyu Gao, Aman Madaan, Shuyan Zhou, Uri Alon, Pengfei Liu, Yiming Yang, Jamie Callan, and Graham Neubig. 2023. PAL: program-aided language models. In *Proceedings of the 40th International Conference on Machine Learning (ICML '23, Vol. 202)*.

[16] Jakob Hahn, Sarah Heiler, Michael B. Kane, Sumee Park, and Werner Jensch. 2022. The Information Gap in Occupant-Centric Building Operations: Lessons Learned from Interviews with Building Operators in Germany. *Frontiers in Built Environment* 8 (2022). doi:10.3389/fbuil.2022.838859

[17] Florimond Houssiau, Luc Rocher, and Yves-Alexandre de Montjoye. 2022. On the difficulty of achieving Differential Privacy in practice: user-level guarantees in aggregate location data. *Nature Communications* 13, 1 (2022).

[18] Zimo Ji, Daoyuan Wu, Wenyuan Jiang, Pingchuan Ma, Zongjie Li, Yudong Gao, Shuai Wang, and Yingjiu Li. 2026. Taming Various Privilege Escalation in LLM-Based Agent Systems: A Mandatory Access Control Framework. arXiv:2601.11893

[19] Changjiang Jiang, Fengchang Yu, Haihua Chen, Wei Lu, and Jin Zeng. 2025. TabDSR: Decompose, Sanitize, and Reason for Complex Numerical Reasoning in Tabular Data. arXiv:2511.02219

[20] Jie Jiang, Riccardo Pozza, Nigel Gilbert, and Klaus Moessner. 2020. MakeSense: An IoT Testbed for Social Research of Indoor Activities. *ACM Transactions on Internet of Things* 1, 3 (2020). doi:10.1145/3381914

[21] Janghyun Kim, Barry Hooper, Tianzhen Hong, and Marc-Antoine Paré. 2022. A review of preserving privacy in data collected from buildings with differential privacy. *Journal of Building Engineering* 56 (2022). doi:10.1016/j.jobe.2022.104724

[22] Evan King, Haoxiang Yu, Sangsu Lee, and Christine Julien. 2024. Sasha: Creative Goal-Oriented Reasoning in Smart Homes with Large Language Models. *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies* 8, 1 (2024). doi:10.1145/3643505

[23] Evan King, Haoxiang Yu, Sahil Vartak, Jenna Jacob, Sangsu Lee, and Christine Julien. 2024. Thoughtful Things: Building Human-Centric Smart Devices with Small Language Models. arXiv:2405.03821

[24] Yunshi Lan, Gaole He, Jinhao Jiang, Jing Jiang, Wayne Xin Zhao, and Ji-Rong Wen. 2021. A Survey on Complex Knowledge Base Question Answering: Methods, Challenges and Solutions. arXiv:2105.11644

[25] Sungmin Lee, Minju Kang, Joonhee Lee, Seungyong Lee, Dongju Kim, Jingi Hong, Jun Shin, Pei Zhang, and JeongGil Ko. 2025. LLM-based Question-Answer Framework for Sensor-driven HVAC System Interaction. arXiv:2507.04748

[26] Mingchen Li and Zhe Wang. 2026. BuildingGPT: Query building semantic data using large language models and vector-graph retrieval-augmented generation. *Building and Environment* 287 (2026). doi:10.1016/j.buildenv.2025.113855

[27] Haohao Luo, Zexi Li, Yuexiang Xie, Wenhao Zhang, Yaliang Li, and Ying Shen. 2026. IntentRL: Training Proactive User-intent Agents for Open-ended Deep Research via Reinforcement Learning. arXiv:2602.03468

[28] Ozan Baris Mulayim, Avia Anwar, Umut Mete Saka, Lazlo Paul, Anand Krishnan Prakash, Gabe Fierro, Marco Pritoni, and Mario Bergés. 2025. BuildingQA: A Benchmark for Natural Language Question Answering over Building Knowledge Graphs. In *Proceedings of the 12th ACM International Conference on Systems for Energy-Efficient Buildings, Cities, and Transportation (BuildSys '25)*. doi:10.1145/3736425.3770097

[29] Ozan Baris Mulayim, Anand Krishnan Prakash, Lazlo Paul, and Marco Pritoni. 2025. Extraction and Analysis of Time Series Data from Building Automation Systems Using Large Language Models. doi:10.20357/B73W4C

[30] Shishir G. Patil, Tianjun Zhang, Xin Wang, and Joseph E. Gonzalez. 2024. Gorilla: Large Language Model Connected with Massive APIs. In *Advances in Neural Information Processing Systems (NeurIPS '24)*. doi:10.52202/079017-4020

[31] Nandita Pattnaik, Shujun Li, and Jason R.C. Nurse. 2024. Security and Privacy Perspectives of People Living in Shared Home Environments. *Proceedings of the ACM on Human-Computer Interaction* 8, CSCW2 (2024). doi:10.1145/3686907

[32] Nandita Pattnaik, Shujun Li, and Jason R. C. Nurse. 2023. A Survey of User Perspectives on Security and Privacy in a Home Networking Environment. *Comput. Surveys* 55, 9 (2023). doi:10.1145/3558095

[33] Elena Planas, Salvador Martínez, Marco Brambilla, and Jordi Cabot. 2022. Towards Access Control Models for Conversational User Interfaces. In *Enterprise, Business-Process and Information Systems Modeling*.

[34] Elena Planas, Salvador Martínez, Marco Brambilla, and Jordi Cabot. 2023. Modeling and enforcing access control policies in conversational user interfaces. *Software Systems Modeling* 22, 6 (2023). doi:10.1007/s10270-023-01131-3

[35] Mohammadreza Pourreza and Davood Rafiei. 2023. DIN-SQL: decomposed in-context learning of text-to-SQL with self-correction. In *Advances in Neural Information Processing Systems (NeurIPS '23)*.

[36] Yujia Qin, Shihao Liang, Yining Ye, Kunlun Zhu, Lan Yan, Yaxi Lu, Yankai Lin, Xin Cong, Xiangru Tang, Bill Qian, Sihan Zhao, Lauren Hong, Runchu Tian, Ruobing Xie, Jie Zhou, Mark Gerstein, Dahai Li, Zhiyuan Liu, and Maosong Sun. 2023. ToolLLM: Facilitating Large Language Models to Master 16000+ Real-world APIs. arXiv:2307.16789

[37] Dmitriy Rivkin, Francois Hogan, Amal Feriani, Abhisek Konar, Adam Sigal, Steve Liu, and Greg Dudek. 2024. SAGE: Smart home Agent with Grounded Execution. arXiv:2311.00772

[38] Md Rashad Al Hasan Rony, Uttam Kumar, Roman Teucher, Liubov Kovriguina, and Jens Lehmann. 2022. SGPT: A Generative Approach for SPARQL Query Generation From Natural Language Questions. *IEEE Access* 10 (2022). doi:10.1109/ACCESS.2022.3188714

[39] Luis Sanchez, Luis Muñoz, Jose Antonio Galache, Pablo Sotres, Juan R. Santana, Veronica Gutierrez, Rajiv Ramdhany, Alex Gluhak, Srdjan Krco, Evangelos Theodoridis, and Dennis Pfisterer. 2014. SmartSantander: IoT experimentation over a smart city testbed. *Computer Networks* 61 (2014). doi:10.1016/j.bjp.2013.12.020 Special issue on Future Internet Testbeds – Part I.

[40] Yuan Sui, Mengyu Zhou, Mingjie Zhou, Shi Han, and Dongmei Zhang. 2024. Table Meets LLM: Can Large Language Models Understand Structured Table Data? A Benchmark and Empirical Study. In *Proceedings of the 17th ACM International Conference on Web Search and Data Mining (WSDM '24)*. doi:10.1145/3616855.3635752

[41] Shayan Talaei, Mohammadreza Pourreza, Yu-Chen Chang, Azalia Mirhoseini, and Amin Saberi. 2024. CHESS: Contextual Harnessing for Efficient SQL Synthesis. arXiv:2405.16755

[42] Yuan Tian, Nan Zhang, Yueh-Hsun Lin, XiaoFeng Wang, Blase Ur, XianZheng Guo, and Patrick Tague. 2017. SmartAuth: User-Centered Authorization for the Internet of Things. In *USENIX Security Symposium*.

[43] Yuchen Wang and Lu Liu. 2024. Research on sustainable green building space design model integrating IoT technology. *PLOS ONE* 19, 4 (2024). doi:10.1371/journal.pone.0298982

[44] Thomas Weng, Anthony Nwokafor, and Yuvraj Agarwal. 2013. BuildingDepot 2.0: An Integrated Management System for Building Analysis and Control. In *Proceedings of the 5th ACM Workshop on Embedded Systems for Energy-Efficient Buildings (BuildSys '13)*. doi:10.1145/2528282.2528285

[45] Tianwei Xing, Luis Garcia, Federico Cerutti, Lance Kaplan, Alun Preece, and Mani Srivastava. 2021. DeepSQA: Understanding Sensor Data via Question Answering. In *Proceedings of the International Conference on Internet-of-Things Design and Implementation (IoTDI '21)*. doi:10.1145/3450268.3453529

[46] Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. 2023. ReAct: Synergizing Reasoning and Acting in Language Models. arXiv:2210.03629

[47] Wen Ye, Wei Yang, Defu Cao, Yizhou Zhang, Lumingyuan Tang, Jie Cai, and Yan Liu. 2026. TS-Reasoner: Domain-Oriented Time Series Inference Agents for Reasoning and Automated Analysis. arXiv:2410.04047

[48] Xiaofan Yu, Lanxiang Hu, Benjamin Reichman, Dylan Chu, Rushil Chandrupatla, Xiyuan Zhang, Larry Heck, and Tajana S. Rosing. 2025. SensorChat: Answering Qualitative and Quantitative Questions during Long-term Multimodal Sensor Interactions. *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies* 9, 3 (2025). doi:10.1145/3749496

[49] Wenqi Zhang, Yongliang Shen, Zeqi Tan, Guiyang Hou, Weiming Lu, and Yueting Zhuang. 2025. Data-Copilot: Bridging Billions of Data and Humans with Autonomous Workflow. arXiv:2306.07209

[50] Zhiping Zhang, Michelle Jia, Hao-Ping (Hank) Lee, Bingsheng Yao, Sauvik Das, Ada Lerner, Dakuo Wang, and Tianshi Li. 2024. "It's a Fair Game", or Is It? Examining How Users Navigate Disclosure Risks and Benefits When Using LLM-Based Conversational Agents. In *Proceedings of the 2024 CHI Conference on Human Factors in Computing Systems (CHI '24)*. doi:10.1145/3613904.3642385

[51] Jiale Zhao, Ke Fang, and Lu Cheng. 2026. When and What to Ask: AskBench and Rubric-Guided RLVR for LLM Clarification. arXiv:2602.11199

[52] Junwen Zheng and Martin Fischer. 2023. BIM-GPT: a Prompt-Based Virtual Assistant Framework for BIM Information Retrieval. arXiv:2304.09333

[53] Haozhe Zhou, Mayank Goel, and Yuvraj Agarwal. 2024. Bring Privacy To The Table: Interactive Negotiation for Privacy Settings of Shared Sensing Devices. In *Proceedings of the 2024 CHI Conference on Human Factors in Computing Systems (CHI '24)*. doi:10.1145/3613904.3642897

[54] Sharif Noor Zisad and Ragib Hasan. 2026. LLMAC: A Global and Explainable Access Control Framework with Large Language Model. In *2026 IEEE 23rd Consumer Communications & Networking Conference (CCNC)*. doi:10.1109/CCNC65079.2026.11366514

APPENDIX

A LLM Prompts

The LLM prompts used by BuildingChat are available at https://doi.org/10.17605/OSF.IO/4J5GQ.

B Formative Study Interview Script

B.1 Introduction and Consent

My name is [interviewer name]. I'm a [title] at [institution]. Thank you for your interest in our research. As a refresher, we are going to have a discussion together about your experiences with office buildings, and talk about some hypothetical scenarios.

Before we get into the interview, we wanted to first go over a consent form with you. Take some time to review the form here: [give participant consent form] Do you have any questions about this form? [Answer questions]

I just want to highlight a few key points in that form. Our conversation will be 60 minutes. I will be asking you some questions about your day-to-day experiences with office buildings, and then we'll dig deeper into some hypothetical scenarios. I will be taking some notes about our discussion in my notebook, and I will also be using [either zoom or device] to record our discussion so that our research team can refer to it later. I will not write down any sensitive information, and we will remove sensitive information from our transcript. However, to the extent that you are able to, please avoid revealing sensitive information in this conversation. When we are done with our conversation, you will receive a $15 Amazon gift card. Do you have any other questions before we begin [Answer questions]

Are you still interested in participating in this interview? Are you okay with me taking notes? Are you okay with the interview audio being recorded? [If agree, start recording]

B.2 Establishing Persona

[All participants except emergency managers] First, I'm going to ask you some questions about your experience with this building and what your relationship is with it. How long have you been coming to this building? Is coming to this building a part of your day-to-day routine? What does a typical day look like for you in this building? What kinds of spaces do you typically occupy within this building? What frustrations do you have with this building and with the spaces that you occupy during your typical day? Some examples might be, "the temperature in my office fluctuates frequently" or "my floor is really loud in the morning"

[Emergency managers] For what community do you manage emergencies for? What kinds of emergencies occur at buildings typically that you manage? When you arrive on scene, what kinds of information do you look for? What would be useful? How do you currently learn about the building information that you need? Would any other information be useful to you?

B.3 Pain Point Deep Dive

[select 1-2 pain points described by the participant. For each pain point, ask the following] You mentioned [pain point]. Let's dive deeper into the challenges that you face with this. How often does [pain point] happen? Are there any specific times, or indicators that [pain point] is more likely to occur? Can you give a specific example of when/how [pain point] happened? Describe the play-by-play of how this happened. Who else seems to have this same problem? Are there ways that you think you experience [pain point] differently than other people in the building?

B.4 Introducing Smart Building Concept

Now, I'd like to explore the concept of having a more "intelligent" building with you. Imagine this building could "know" things about how the spaces and facilities are being used, and share that information with the occupants.

The building's intelligence is informed by sensors placed around the building. These sensors are able to capture information such as loudness and sounds (but not speech), temperature, and motion. There are multiple sensors in each room of the building, as well as public spaces.

Imagine that this system has the following boundaries: (1) for privacy, the system can't track specific individuals, or know who you are, (2) it filters raw audio data to remove speech and cannot identify specific people. To help solidify this building concept, I will provide the following example: Two sensors are placed on the walls and ceiling, and can detect changes in the lighting, temperature, and speech. You can ask the question "what room has the brightest lighting?", and the system may answer, "room 345". You can ask the question, "how productive was I from 3-4", then the system may answer "you worked for 45 minutes in the past hour"

[All participants] Imagine that you are going through your typical interactions with this building. What would make the day go smoother?

[All participants] Imagine that you could ask the building questions through an app. What kinds of questions would you want to ask?

[If emergency manager] In time-sensitive scenarios, would you want to use an app such as this? What kinds of information would be useful to access?

[All participants] What (if anything) would you want the building to tell you proactively?

[If building occupant, administrator, or facilities staff member] How might your daily routine change if your building was "smart"? How might a smart building make you more productive/comfortable using the spaces that you have access to? If you were a visitor in a new building, what kinds of information would you want to know? What building insights would be useful to you?

[If the person has their own space in the building] As someone who has your own space here, what would you want to know about your specific space? What would you want to know about the building beyond your own space?

[If the person is an administrator] What kinds of responsibilities do you hold/what does your job entail? From your experience managing [department/resource], what building insights would help your decision-making?

[If the person is a facilities staff member] What kinds of responsibilities do you hold/what does your job entail? From your experiences maintaining/supporting this building, what would help you be more effective?

B.5 Exploring Boundaries and Privacy Concerns

Now, I'm going to ask you some questions about what you would NOT want from a smart building.

What concerns would you have in general with a smart building system? Based on our discussion so far, what parts of a smart building system would make you uncomfortable or uneasy? What kinds of information would you be ok with the building knowing about you versus not? Would you want control over when and how the building interacts with you? What would this control look like?

B.6 Wrap Up

Thank you very much for participating in our discussion today! Just a few last questions as we wrap things up: Do you have anything else that you would like to tell me? How was this conversation for you? Did anything that we discussed surprise you? Are there others that we should talk to who have different experiences with your building? As a reminder, we will follow up with the compensation and send you the gift card as soon as possible.

C Template Library Details

Table 7 summarizes the six templates in BuildingChat's library. This appendix describes each template's function and the formative study pattern it addresses. `current_measurement` addresses point-in-time queries (e.g., "What is the temperature on floor 3?"). It extracts current values from sensor data with configurable aggregation and produces a `measurement` packet. `spatial_compare` addresses cross-location queries (e.g., "Which floor is warmer?"). It takes `measurement` packets from multiple locations and produces an `analysis_result` with rankings and margins. `temporal_analysis` addresses trend queries (e.g., "How has the temperature changed today?"). It analyzes time-series data to identify directional changes, peaks, and statistical significance, producing both a `time_series` and an `analysis_result`. `threshold_filter` addresses conditional queries (e.g., "Which rooms are above 75°F?"). It filters locations based on whether measurements meet specified criteria, producing a `location_set`. `web_search` retrieves external context such as current weather conditions. Unlike the analytical templates, it takes no sensor input packets and produces an `analysis_result` directly. For queries like "Is it warmer inside than outside?", the DAG contains parallel branches: one processing building sensors and one retrieving external data, with the response generator synthesizing both into a coherent answer. `info_lookup` handles informational queries about the building itself, such as room capacities, operating hours, or available sensor modalities. Like `web_search`, it takes no sensor input packets and routes through building-specific documentation.

D Test Query Generation Details

This appendix describes the test query generation procedure in detail.

D.1 Generation Procedure

We generate each test case in three phases: parameter sampling, ground truth computation, and natural language query generation.

Phase 1: Parameter Sampling. For each query, we sample structured parameters from the mock building's validated topology. We draw location tags from predefined combinations that exist in the building (e.g., `{floor: 3, locationType: office}`, `{location: 310}`). We sample sensor modalities from the 11 available types (temperature, humidity, light, microphone, barometer, accelerometer, color, magnetometer, grideye, motion, wifi). We sample time specifications in one of five formats: relative point (e.g., "2 hours ago"), relative range (e.g., "over the last 3 hours"), absolute point (e.g., "at 14:30 on Jan 5"), absolute range (e.g., "from 14:00 to 18:00 on Jan 5"), or current ("right now"). We validate that all sampled times fall within the mock data range. For comparison queries, we sample a second location-time pair, ensuring the two operands differ meaningfully (different location or at least two hours apart). For filter queries, we additionally sample a threshold value and comparison operator.

Phase 2: Ground Truth Computation. We compute the ground truth answer deterministically from the mock data using manually constructed procedures for each query type. For measurement queries, we compute a sensor-weighted mean across all spaces matching the location tags. For comparison queries, we compute the aggregated value for each operand and determine which is higher or lower. For filter queries, we count spaces whose average sensor value meets the threshold condition. For trend queries, we fit a linear regression over the time window and report the direction (increasing, decreasing, or stable) along with statistical significance. We discard and resample test cases (up to 5 attempts) where the ground truth is missing or ambiguous. For comparison queries, we additionally discard cases where the two values differ by less than 10% of the maximum, avoiding near-equal comparisons that would be ambiguous to evaluate.

Phase 3: Natural Language Generation. We use two strategies depending on query type. For boolean comparison queries (e.g., "Is the temperature in room 310 warmer than in room 350?"), we use deterministic templates that

combine LLM-generated operand descriptions with sensor-appropriate comparison phrases (e.g., “warmer than” for temperature, “stronger than” for WiFi signal). This avoids the risk of an LLM inverting the comparison direction, which would make the ground truth label incorrect. For all other query types, we prompt the LLM (Qwen-3 235B) with the sampled parameters to generate the query text. Each prompt includes the operation type, sensor display name, location description, time specification, and expected tags. The prompts enforce several constraints to ensure query quality: (a) queries must use exact sensor names rather than vague paraphrases (e.g., “PIR motion reading” rather than “motion activity”), (b) time references must be unambiguous (ranges must specify both start and end), (c) the LLM must not invent location details beyond what appears in the sampled tags, and (d) queries must request measurable quantities rather than yes/no formulations (for measurement queries). Each prompt includes 4–6 worked examples demonstrating correct formatting. We manually review all generated query texts and adjust them to remove remaining ambiguities.

D.2 Representative Prompt

Below is the prompt template we use for measurement queries. Other query types (comparison, trend, filter) follow the same structure with type-specific instructions and examples.

```
Generate a natural language query for a building
sensor system.

Context:
- Operation: MEASURE (single measurement query)
- Sensor type: {sensor_display}
- Location: {location}
- Time: {time}
- Expected tags: {tags}

The query should:
1. Clearly specify the location
2. Clearly specify the sensor using the EXACT name
3. Clearly specify the time
4. Be natural and conversational
5. Use ONLY location information from the provided
   Tags — do NOT invent details
6. Phrase as a numeric measurement question, NOT a
   yes/no question

[4-6 worked examples omitted for brevity]

Generate a single, clear query question. Do not
include any explanation, just the query itself.

Query:
```

D.3 Example Test Cases

Table 8 shows representative generated test cases across the four primary query categories.

E Design Iteration Study Interview Script

This script was used for the initial interview study which informed further design iteration (Section 4.7).

Table 8. Example test cases from the evaluation set, showing sampled parameters, computed ground truth, and generated natural language query for each category.

Category	Parameters	Time Spec	Ground Truth	Generated Query
Measurement	sensor: temperature, tags: {floor: 3}	relative range: past 1 hour	72.4°F (mean)	"What was the average temperature on floor 3 over the past hour?"
Comparison	sensor: humidity, $tags_1$: {location: 310}, $tags_2$: {location: 350}	both: current point	Location 1 higher (61.2% vs 54.8%)	"How does the humidity in room 310 right now compare to the humidity in room 350 right now?"
Filter	sensor: temperature, tags: {floor: 2}, threshold: > 75°F	relative range: past 3 hours	4 rooms	"How many rooms on Floor 2 had an average temperature above 75°F over the past 3 hours?"
Trend	sensor: light, tags: {locationType: office, cardinality: NE}	relative range: today	increasing ($p < 0.05$)	"What's the light level trend in NE offices today?"

E.1 Introduction and Consent

My name is [interviewer name]. I'm a [title] at [institution]. Thank you for your interest in our research. As a refresher, we are going to have a discussion together about our building chatbot called BuildingChat.

Before we get into the interview, we wanted to first go over a consent form with you. Take some time to review the form here: [give participant consent form] Do you have any questions about this form? [Answer questions]

I just want to highlight a few key points in that form. Our conversation today will be 60 minutes. In this discussion, we want you to test out our app, BuildingChat, which enables users to ask questions of their building using natural language. We will set up the BuildingChat app, explore the questions that you might have using the BuildingChat app, and ask you to generate more questions using theoretical scenarios or situations. We'll then ask you to describe your experience with using the app. You will be compensated $20 for your time today.

After our conversation, you will have the opportunity to use BuildingChat for a week as much as you wish.

Finally, we will set up a 30-minute interview for you to tell us how your experience was with the week-long usage of BuildingChat. You will be compensated $10 for your time.

For our conversation today, I will be taking some notes about our discussion in my notebook, and I will also be recording our discussion so that our research team can refer to it later. I will not write down any sensitive information, and we will remove sensitive information from our transcript. However, to the extent that you are able to, please avoid revealing sensitive information in this conversation. Do you have any other questions before we begin [Answer questions]

Are you still interested in participating in this interview? Are you okay with me taking notes? Are you okay with the interview audio being recorded? [If agree, start recording]

E.2 Establishing Persona

First, I'm going to ask you some questions about your experience with this building and what your relationship is with it. How long have you been coming to this building? Is coming to this building a part of your day-to-day routine? What does a typical day look like for you in this building? What kinds of spaces do you typically occupy within this building?

E.3 Set up BuildingChat

Now, we're going to have you install our BuildingChat app. BuildingChat is an application that allows you to ask the building questions in natural language, and receive responses based on sensor-based insights. [Direct participant to log into webapp and create account]

Let me go through a few more points of information of what kinds of sensor-insights are available in our building, and give you an example of what kinds of questions you can ask. [Participant is shown flyer with information about sensors installed in building]

With BuildingChat, you can ask questions based on the insights from these sensors. For example, you can see that there are temperature sensors available, so a question that I can ask BuildingChat is "How hot is my office?" Additionally note that for this study, we are using Qwen 3. We are running Qwen3 on the cloud. Note that no sensor data is sent to the cloud. In our final deployment, we plan to run the models locally. Note that BuildingChat will also tell you when it needs more information to answer a question, if you do not have authorization to access the information, or if the information doesn't exist. In these scenarios, the system will tell you why it cannot provide this information. It will additionally help you refine your query to a related query that is authorized or that has information available.

E.4 Walking Through Imagined Scenarios

Now, we will explore some sample questions that might come that can be answered by BuildingChat. Now, with these questions, try asking BuildingChat a few questions. We'll be here to help, but please think aloud about any responses that you get from BuildingChat, and how you formulate your queries.

This is a more rudimentary chatbot than most modern LLM chatbots, but it is the first to our knowledge that utilizes sensor insights from buildings to answer questions in natural language. So, please be a bit patient with the system! You can have back and forth interactions with the chatbot. We recommend that if you are asking follow up questions, or questions about the same topic, you can use one chat, but if you decide to switch topics, we recommend using a new chat. For example, if you are having a discussion with BuildingChat about the temperature of your office, and want to switch to asking about the humidity of a conference room, we recommend that you use a new chat for the second query.

[Interviewer selects five scenarios to discuss (Appendix G), with three scenarios known to be supported by BuildingChat, one scenario that requests inaccessible private data, and another scenario that requests data that is not available. For each scenario, ask participant to enter questions into BuildingChat and explain their reactions as they review response and follow-ups as appropriate.]

E.5 Overall reactions and conclusion

For each query that you asked, did you feel that BuildingChat's response contained the information that you wanted from this query? If BuildingChat asked a clarification question, did you feel that you were able to easily understand why you needed to clarify the question that you initially asked? Were you able to understand how to clarify the question that you initially asked?

How was this experience for you? What did you learn about your space and your building? What felt challenging about using our tool? Would you recommend using this tool to other people in the building?

Thank you very much for participating in our discussion today! Just a few last questions as we wrap things up: Do you have anything else that you would like to tell me? As a reminder, we will follow up with the compensation and send you the gift card as soon as possible. Feel free to email us for any more clarifications and questions that you might have!

F Usability Evaluation Interview Script

This script was used for the interview study to evaluate the usability of BuildingChat (Section 6).

F.1 Introduction and consent

My name is [interviewer name]. I'm a [title] at [institution]. Thank you for your interest in our research. As a refresher, we are going to have a discussion together about our building chatbot called BuildingChat.

Before we get into the interview, were you able to fill the consent form? If not, please take a moment to review and sign the consent form now. Did you have any questions about this consent form? [Answer questions]

Our conversation today will be 1 hour long. The goal of this interview is for you to tell us how your experience was with the three day-long usage of BuildingChat. You will be compensated $30 for your time.

For our conversation today, I will be taking some notes about our discussion in my notebook, and I will also be using zoom to record our discussion so that our research team can refer to it later. I will not write down any sensitive information, and we will remove sensitive information from our transcript. However, to the extent that you are able to, please avoid revealing sensitive information in this conversation. Do you have any other questions before we begin? [Answer questions]

Are you still interested in participating in this interview? [**Yes**/No → if yes continue] Are you okay with me taking notes? [Yes/No → if yes continue, if no, say "that's ok, we can proceed without the note-taking" (and then proceed without note taking)] Are you okay with the interview audio being recorded? [Yes/No → if yes continue] Ok, I am going to start recording. As a reminder, please do not share anything private or sensitive about yourself or anyone else. [turn on recording] Ok, we are recording now.

F.2 Establishing persona

First, I'm going to ask you some questions about your experience with this building and what your relationship is with it. How long have you been coming to this building? Is coming to this building a part of your day-to-day routine? What does a typical day look like for you in this building? What kinds of spaces do you typically occupy within this building?

[As you discuss, ask follow up questions for clarifications]

F.3 Questions about overall experience and insights

First, we're going to have you look back at your past 3 days. How much did you use BuildingChat?

[if used system at least once] Check your previous chats to look at what you've asked before.

How was your overall experience using it? What kinds of questions did you find yourself asking? Were you more curious about a specific type of space like huddle rooms? What did you learn? How did you come across this information (i.e, which kinds of queries revealed this information)?

[if used system not at all] Are there any specific reasons you did not use it?

F.4 Using BuildingChat for specific scenarios

Interviewer selects three scenarios to discuss, with one scenarios known to be supported by BuildingChat, one scenario that requests inaccessible private data, and another scenario that requests data that is not available. For each scenario, ask participant to enter questions into BuildingChat and explain their reactions as they review response and follow-ups as appropriate.

F.5 Using BuildingChat for user-generated questions

Now, we'd like you to ask questions of your own choosing. Can you open it up on your computer and think about what else you might want to ask? We'll have you ask 2-3 questions.

[Participant enters questions into BuildingChat, approx. 5 minutes]
[if no ideas, then suggest following]:
Try to think of a question that you would like to ask BuildingChat. To give you some ideas of questions, try to think about the following prompts:
Imagine that you are going through your typical interactions with this building. What would make the day go smoother?
Imagine that you could ask the building questions through an app. How would your interactions change?
[If they still don't have ideas, suggest something from list of questions]

F.6 Questions about usability

Now, we're going to have you answer questions about your overall experience using BuildingChat.

- Did you feel that BuildingChat was easy to use and intuitive?
- Was there a learning curve?
- What was your process of learning how to use BuildingChat?
- Did you feel that the responses that you got from BuildingChat were reliable? [if no] Can you recall a question where you felt that it wasn't reliable and explain why you felt that it wasn't reliable?
- Did you feel that the information that you received was sufficient?
- Did you feel that BuildingChat answered your questions with the appropriate level of detail?
- Was the response understandable and useful? Were you able to utilize the information?

Now, we're going to ask you about the quality of the responses that you received.
Did you feel that, overall, the intention of your question was well understood? [If Understood] Can you recall an example of instances where your intention was well understood, [if not understood] Can you recall an example of instances when it was not
Did you encounter scenarios where BuildingChat suggested an alternative query? Did you feel like the clarification for the query change was understandable? Did you understand why the clarification was needed?

F.7 Questions about scenarios where BuildingChat cannot answer questions

When you have asked questions to building chat, were there any questions it said it couldn't answer because the system doesn't have the ability (either knowledge about the building or sensor information) to answer it? [If yes]

- Do you remember which questions? And did you expect it would work?
- Do you think the system should be able to answer this type of question?

[if No] continue…
When you have asked questions to building chat, were there any questions it said it couldn't answer because you had unauthorized access? [If yes]

- Do you remember which questions?
- Did you expect that you would not be authorized to access that information?
- Do you think that policy is appropriate?

[if no, they think is appropriate] Would you want other people to be able to ask that question?
Are there any things you want to ask building chat that you don't think it can do yet?
What features do you wish you had as a part of building chat? What features would benefit building chat and you as a user?
Please fill out this final survey on your overall user experience of BuildingChat!

F.8 SUS survey

Survey is completed on laptop. All questions are on 5-point Likert scale, strongly disagree to strongly agree

- I think that I would like to use this system frequently
- I found the system unnecessarily complex
- I thought the system was easy to use
- I think that I would need the support of a technical person to be able to use this system
- I found the various functions in this system were well integrated
- I thought there was too much inconsistency in this system
- I would imagine that most people would learn to use this system very quickly
- I found the system very cumbersome to use
- I felt very confident using the system
- I needed to learn a lot of things before I could get going with this system

F.9 Wrap-up

Thank you very much for participating in our discussion today! Just a few last questions as we wrap things up: Do you have anything else that you would like to tell me? As a reminder, we will follow up with the compensation and send you the gift card in the next few weeks. Feel free to email us for any more clarifications and questions that you might have!

G Usability Evaluation "Hypothetical Scenarios"

This appendix includes the "hypothetical scenarios" for the design iteration study (Section 4.7) and usability evaluation (Section 6).

We first created a list of 41 questions that one could ask BuildingChat. We generated this list from the formative study (described in Section 3), as well as questions generated by the research team. The questions were designed to cover a diverse range of intentions and sensor types. We additionally included questions that would violate privacy boundaries and questions that would involve data that does not exist.

From this list of questions, we generated a list of hypothetical scenarios by prompting OpenAI's GPT-5 model to write a hypothetical scenario around each prompt. We then adjusted the imagined scenarios to remove hallucinated information or context. The scenarios that we utilized for users were as follows:

Scenario	Expected Response
imagine that it's a cold winter morning, you're feeling sluggish at your desk, and you want to relocate to a brighter, warmer spot in the building to work more comfortably.	Allowed
imagine that you're about to join an important video call and need to quickly find a private room that isn't already occupied.	Allowed
imagine that you've just arrived at the office for the day and want to choose a workspace that best fits your tasks, whether that's quiet focus or collaboration.	Allowed
imagine that your office feels unusually warm and you're trying to figure out whether this is a building-wide issue or something specific to your room.	Allowed
imagine that you need to have a quick in-person discussion with collaborator X and want to know if they're currently in the building.	Private data
imagine that you're sensitive to odors and are deciding whether to spend the afternoon working on floor 3.	No data
imagine that it's lunchtime and you want to heat up your food without waiting in line at a busy microwave.	No data
imagine that you're planning your day and want to avoid areas that might be closed due to maintenance or unexpected issues.	No data

imagine that you're about to move into room 347 for a meeting and want to check if the temperature will be comfortable.	Private data
imagine that you see unusual activity near Amanda's office and want to know if there's a meeting, event, or issue going on.	Private data
imagine that you want a general overview of current building activity.	Allowed
imagine that you're looking for a quiet place to focus and want to avoid the noisiest floor in the building.	Allowed
imagine that you're visiting the office for the first time and need directions to find X person's office quickly.	No data
imagine that you're scheduling a meeting and want to check whether your favorite conference rooms are free.	Allowed
imagine that you're standing outside the TCS conference room and want to know if it's currently occupied before entering.	Allowed
imagine that you're feeling uncomfortable at your desk and want confirmation on whether your office temperature is above normal.	Allowed
imagine that your friend's office is office 347 and you have noticed it feels cold every time you're there.	Private data
imagine that you spend a lot of time working in the lab and want to know if its low temperature is a consistent pattern.	Allowed
imagine that you're trying to avoid peak crowd times and want to know when the building is usually busiest.	Allowed
imagine that you're feeling dehydrated and want to know whether there's cold drinking water near area X.	No data
imagine that you're planning focused work and want to confirm whether a specific space is quiet at a certain time.	Allowed
imagine that you want to avoid distractions and are trying to identify when space X tends to get loud.	Allowed
imagine that you're looking for underutilized areas to work and want to know which spaces are already heavily used.	Allowed
imagine that you want to plan meetings during less busy times and need to know peak occupancy periods.	Allowed
imagine that your team needs a quick informal meeting and you're trying to find an available huddle space nearby.	Allowed
imagine that you're new to the building and want to know if there are small collaboration spaces available.	Allowed
imagine that you often work late and want to understand how the building is typically used after normal office hours.	Allowed
imagine that you're reflecting on your work habits and want insight into how much time you actually spend at your desk.	Allowed
imagine that you're deciding whether to step outside for a break and want to compare outdoor conditions to the indoor temperature.	Allowed
imagine that you're monitoring comfort conditions and want to know if the NE corridor has been gradually heating up.	Allowed
imagine that you're cold after sitting still for hours and want to relocate to the warmest area on floor 3.	Allowed
imagine that you noticed temperature fluctuations while walking through the corridor and want to understand whether it's been consistently warm or cold recently.	Allowed
imagine that you're planning to work in the NE corridor area on floor 3 and want to know how comfortable the temperature is right now.	Allowed
imagine that the air feels unusually dry or sticky and you want to check the current humidity level.	Allowed
imagine that you're planning a task that requires good lighting and want to confirm the brightness in room 347.	Private data
imagine that you're considering using that conference room for a call and want to know if noise will be an issue.	Allowed
imagine that you're adjusting your clothing layers and want to know the exact temperature at your desk.	Allowed
imagine that you're reviewing a reported disturbance and want to understand how loud the corridor was at that time.	Allowed
imagine that facilities management asked you to check whether temperatures were within range earlier in the day.	Allowed
imagine that you're analyzing historical lighting conditions in 347 on Jan 2nd	Allowed
imagine that you're trying to figure out the average temperature in the NE corridor over the last hour.	Allowed

Table 9. Scenarios discussed in interview