

Article

An AI Answer-Validation Method Using Agentic RAG for Datasheet Inquiry for IoT Application System Deployment

Dezheng Kong , Nobuo Funabiki * , Htoo Htoo Sandi Kyaw, I Nyoman Darma Kotama  and Zihao Zhu

Department of Information and Communication Systems, Okayama University, Okayama 700-8530, Japan

* Correspondence: funabiki@okayama-u.ac.jp

Abstract

Internet of Things (IoT) application systems are increasingly adopted in factories, shops, offices, and governments. However, building such systems using various devices and modules remains difficult for non-experts, because they must confirm specifications, communication interfaces, voltage ranges, and operating conditions from technical datasheets before connecting devices. In previous studies, we have explored a *generative AI*-based answering tool for datasheet inquiry using *Retrieval-Augmented Generation (RAG)* for technical guidance of IoT application system deployment. However, the adopted top-*k* RAG pipeline often retrieves multiple related text chunks, which can cause the AI to confuse technically different specifications, such as power output voltage, signal output voltage, and input voltage range, and produce inaccurate answers. In addition, the AI may generate a hallucinated answer if the datasheet does not provide sufficient source information. In this paper, we propose an *AI answer-validation method using agentic RAG for datasheet inquiry* for IoT application system deployment. The method organizes datasheet information into structured specification data, including device models, field types, values, units, conditions, and source information. For question-answering, the agent coordinates structured fact query, top-*k* text retrieval, source checking, and rule-based compatibility comparison according to the question type. Instead of fully relying on the *LLM* to interpret retrieved chunks, this method adopts structured specifications and deterministic source checks before accepting the final answer. For evaluation, we constructed a dataset from 20 IoT datasheets, including 1000 question-answering tasks with three difficulty levels. Compared with conventional top-*k* RAG, the proposed method improved the correct answer rate from 0.686 to 0.958 for easy questions, from 0.549 to 0.969 for medium questions, and from 0.273 to 0.613 for hard questions, which confirms the effectiveness of the proposed method.

Keywords: agentic RAG; IoT application system; Internet of Things; datasheet question-answering; device compatibility



Academic Editor: Kewei Sha

Received: 9 July 2026

Revised: 8 August 2026

Accepted: 17 August 2026

Published: 19 August 2026

Copyright: © 2026 by the authors.

Licensee MDPI, Basel, Switzerland.

This article is an open access article distributed under the terms and conditions of the [Creative Commons Attribution \(CC BY\)](https://creativecommons.org/licenses/by/4.0/) license.

1. Introduction

In recent years, Internet of Things (IoT) application systems are increasingly deployed in factories, shops, offices, and governments. A variety of new IoT devices and modules have become easier to obtain through the internet. Then, they are widely used in education, prototyping, automation, environmental monitoring, and small-scale embedded systems. As a result, students and novice engineers will purchase sensors, relays, communication modules, gateways, and development boards at low cost to implement an experimental IoT application system. However, building an IoT application system is still not a simple plug-and-play task. Even a small system often requires users to connect several devices

after checking electrical specifications, selecting suitable communication interfaces, and understanding how data is transmitted from local devices to gateways, cloud platforms, and user applications.

Several studies have pointed out that IoT application systems involve various connected devices, context-aware operations, different communication protocols, and complex hardware and software components [1–8]. In practice, this complexity becomes a major obstacle for non-expert users. Even a user who wants to build a small IoT application system often needs to read several datasheets to find answers to basic but important questions, such as whether a sensor can be powered by a given module, whether its output signal can be accepted by another device, or whether the required communication interface is supported. These questions become more difficult for beginners because datasheets are usually written for skilled engineers, assuming prior knowledge of electronics and embedded systems.

Figure 1 illustrates the complexity of an IoT application system deployment from device modules to user applications. This figure illustrates a typical deployment flow that consists of device modules, a gateway or an edge node, a network path, a cloud platform, and user applications. It also highlights that the successful deployment requires compatibility checking across several datasheet-related constraints, which include electrical matching on signal and input characteristics; protocol matching on pins, addresses, and baud settings; connection setup on wireless mode; and data meaning on unit and range interpretation. This structure indicates that IoT application system construction is not only a matter of connecting hardware components, but also of confirming that devices can interoperate correctly across communication, configuration, and data interpretation stages.

This layered structure can be consistent with broad studies on cloud–IoT integration, edge computing, and fog computing, which show that practical IoT application systems often distribute sensing, processing, storage, and application functions across device, edge, gateway, and cloud resources [9–11].

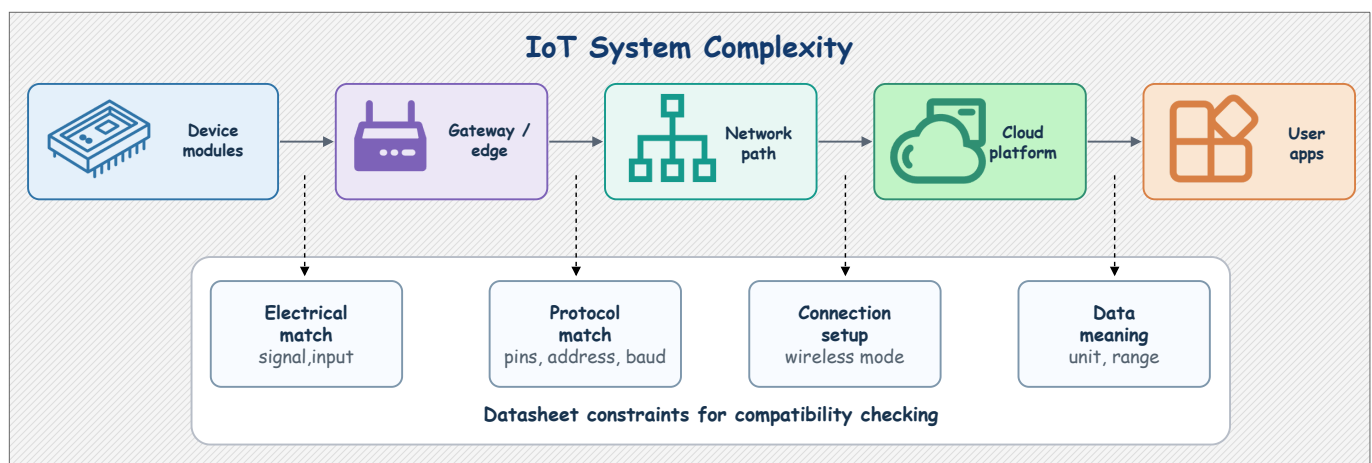


Figure 1. IoT We have added an explanation of the arrows and colors to the figure caption. application system complexity in deployment. Arrows indicate the deployment flow, while colors are used only for visual distinction.

Because of this complexity, device compatibility checking is a very important step in IoT application system construction and deployment. Compatibility is not determined by a single specification. Instead, it must consider several technical constraints, such as supply voltage, signal type, signal voltage range, communication interface, and operating condition. For example, a sensor module may provide an analog output, whereas another module may accept input signals only within a limited voltage range. A gateway may

support wireless communication, but the local device may require a specific serial, analog, or digital interface. Therefore, users must find the corresponding fields correctly in technical documents and compare them under valid operating conditions.

Datasheets contain the important specifications of devices such as supply voltage, signal output voltage, communication interface, operating conditions, electrical limits, and input voltage range. However, the information is often scattered across tables, pin descriptions, electrical characteristics, example circuits, and notes in a datasheet. Besides, for non-expert users, it is difficult to distinguish the fields that look similar but have different technical meanings, such as power supply voltage, signal output voltage, and allowable input voltage range. This problem becomes more serious when users need to judge whether two devices can work together, because compatibility checking requires not only finding the relevant text but also comparing the correct fields. This creates a broader challenge in understanding technical tables and semi-structured document layouts, where values must be interpreted together with headers, row labels, and surrounding context [12,13].

Fortunately, recent progress in document understanding and question-answering has improved the ability of AI models to process visually and structurally complex documents [14–17]. In particular, *large language models (LLMs)* and *Retrieval-Augmented Generation (RAG)* have been widely studied and used for knowledge-intensive question-answering, because retrieval can provide external evidence instead of relying only on model knowledge [18,19]. Nevertheless, neural-based systems may still produce unsupported or technically incorrect answers when the retrieved context is incomplete, ambiguous, or misleading [20,21]. IoT datasheet inquiry is not solved by locating relevant document regions alone. A retrieved region may contain a correct numerical value, but the value can still be used incorrectly if its specification type, unit, condition, or source section is not verified. Therefore, the system must determine whether a retrieved value corresponds to the correct technical field and whether it can be used for a specific setup or compatibility judgment.

A conventional top- k RAG pipeline usually retrieves several text chunks that are semantically related to a user question and asks an LLM to generate the answer. This strategy is useful for general question-answering, but it is not sufficient for handling IoT datasheets. A retrieved chunk may contain several voltage values, where they may refer to the absolute maximum rating, recommended supply voltage, digital input level, analog output range, or example circuit behavior. If they are not distinguished explicitly, the generated answer can be wrong in technical meaning.

Another limitation is that the top- k RAG pipeline does not automatically request additional evidence. Even if the datasheet does not provide enough information, the LLM still generates a plausible answer. For compatibility-related questions, this behavior is highly risky because a reliable answer requires evidence-grounded extraction from both devices, unit normalization, field-type validation, and rule-based comparison. Recent work on tool-using and *agentic LLMs* suggests that LLMs can be used to coordinate reasoning and external actions instead of directly producing answers from a single prompt [22–24]. This motivates a controlled AI answer-validation method for datasheet inquiry in IoT application system deployment.

In this paper, we propose an *AI answer-validation method using agentic RAG-to-datasheet inquiry* for IoT application system deployment. The goal of this method is to support users, especially beginners, in understanding device specifications and checking compatibility during IoT application system construction. Instead of directly asking an LLM to return an answer with several retrieved chunks, the proposed method organizes the datasheet information into structured device facts and uses the agent to coordinate retrieval, extraction, validation, and comparison steps.

The term *agentic RAG* refers to a constrained question-answering workflow in which a language model selects the process according to the question type, and coordinates retrieval, structured specification extraction, source checking, and rule-based comparison. The agent is not used as an unrestricted autonomous decision maker. Instead, we limit its role to process selection and tool coordination, while technical acceptance is controlled by the specification-type validation and deterministic comparison rules.

In alignment with the goal of building a practical assistance tool for constructing an IoT application system, this paper makes the following contributions:

- We design a *structured specification representation* for IoT datasheets. It records device model, specification name, value, unit, condition, source section, and supporting evidence, allowing technically similar fields such as supply voltage, signal output voltage, and allowable input voltage range to be stored and checked separately.
- We propose a *constrained agentic RAG-based answer-validation workflow* for IoT datasheet inquiry. The workflow uses an *LLM* to route questions and coordinate structured fact access, text retrieval, source checking, and rule-based comparison before accepting final answers.
- We introduce *evidence-based validation and compatibility rules* for datasheet question-answering. These rules check specification type, unit, source section, and cross-field comparability before accepting factual answers or compatibility judgments.
- We construct a manually verified IoT datasheet question-answering benchmark and evaluate the proposed method through comparison with a conventional top-*k* RAG baseline.

For evaluation, we constructed a dataset from 20 IoT datasheets, including 1000 question-answering tasks with three difficulty levels. The experimental results show that the proposed method improves answer correctness from 0.686 to 0.958 for easy questions, from 0.549 to 0.969 for medium questions, and from 0.273 to 0.613 for hard questions compared with conventional top-*k* RAG. These results indicate that the IoT datasheet question-answering benefits from structured device facts, specification validation, evidence checking, and rule-based technical comparison in addition to text retrieval.

The remainder of this paper is organized as follows. Section 2 reviews related studies on IoT device complexity, document understanding, RAG, and agentic reasoning. Section 3 defines the datasheet question-answering task and discusses the limitations of conventional top-*k* RAG. Section 4 presents the proposed AI answer-validation method using agentic RAG. Section 5 describes the implementation of the proposed method. Section 6 shows the evaluation of the proposed method. Section 7 discusses the findings and limitations. Finally, Section 8 concludes the paper.

2. Related Work

In this section, we review related studies on IoT device complexity, document understanding, retrieval-augmented question-answering, tool-mediated language-model reasoning, and evidence-grounded answer verification.

2.1. Technical RAG for Document QA

Retrieval-Augmented Generation (RAG) has been widely studied as a method for combining language models with external document retrieval. In previous studies, retrieval was introduced into language-model pre-training to improve open-domain question-answering [25]. Other studies proposed dense retrieval methods for selecting relevant passages from large document collections [26]. Another line of work showed that generative models can aggregate information from multiple retrieved passages when answering open-domain questions [27].

Lyu et al. proposed an iterative planning-and-answering method for knowledge-intensive generation, in which a generated plan guides the repeated selection of fine-grained passages during answer generation [28]. Baek et al. proposed a selective retrieval method that uses internal language-model representations to determine whether further document retrieval is necessary for a given query [29]. However, these studies primarily address how a system plans retrieval or decides whether additional retrieval is needed. They do not specifically address the datasheet-specific problem of verifying whether a selected value has the required specification type, unit, operating condition, and supporting source. The present method therefore focuses on validating retrieved or extracted datasheet facts before they are accepted in an answer.

Knollmeyer et al. used a document knowledge graph to support question-answering in the manufacturing domain [30]. This study processes domain documents at the document level, whereas our presented method validates individual datasheet facts before they are used in an answer.

2.2. Datasheet-Oriented Information Extraction

Information extraction has been studied as a way to identify entities, attributes, and relations from unstructured or semi-structured text. Previous studies showed that contextualized language representations can improve many natural-language understanding tasks, including information extraction and question-answering [31]. Other studies further showed that pre-training on scientific text can improve performance on scientific natural-language processing tasks [32].

Technical documents are difficult to process because their meaning depends not only on text but also on layout, tables, and section structure. Previous document understanding studies combined textual, visual, and layout information to process visually structured documents [33]. More recently, industrial-domain research performed the document-level entity and relation extraction to construct a knowledge graph from industrial documents [34]. This direction is relevant to datasheet processing because datasheets contain tables, pin descriptions, electrical characteristics, and operating conditions.

However, most document understanding methods focus on extracting text, layout elements, or general information units, while IoT datasheet inquiry requires extracted values to be assigned to correct technical fields. In this paper, the target is not only to extract values from datasheets, but also to verify their specification type, unit, condition, and source evidence before accepting an answer.

2.3. Tool-Mediated LLM Reasoning

Recent studies have explored the use of *large language models (LLMs)* as controllers that can select tools, call external modules, and combine intermediate results. Some previous studies proposed modular systems in which *LLMs* are connected with external knowledge sources and discrete reasoning modules [35]. Other studies showed that they can decompose problems into executable programs while delegating precise computation to an external interpreter [36]. Huang et al. further studied an agentic tool use by retrieving relevant tools and using them during multi-step task execution [37].

These studies suggest that language models do not need to directly produce final answers in a single generation step. Instead, they can coordinate tools and intermediate reasoning processes. This idea is related to *agentic RAG*, where retrieval is only one part of a larger process involving planning, tool execution, verification, and answer synthesis. For IoT datasheet inquiry, such a design is important because compatibility questions require multiple operations: retrieving relevant datasheet chunks, extracting fields from two devices, checking source sections, normalizing units, and applying comparison rules.

Unlike general-purpose tool-using systems, the proposed method uses a constrained process specialized for IoT datasheets. The language model is responsible for selecting the appropriate process according to the question type, while technical judgment is controlled by structured facts, source checking, and predefined compatibility rules. This reduces the risk of unsupported or technically confused answers.

2.4. Evidence-Grounded QA and Verification

Evidence-grounded question-answering and fact verification have been studied to reduce unsupported generation and improve answer trustworthiness. Previous studies introduced large-scale fact verification tasks in which claims must be classified based on textual evidence [38]. Other studies extended this idea to scientific claims, requiring systems to retrieve scientific evidence and identify rationales that support or refute a claim [39]. These studies show that answer correctness should be evaluated together with evidence retrieval and rationale quality.

However, *IoT datasheet inquiry* requires a more domain-specific form of evidence validation. It is not enough to cite a retrieved chunk if the cited value is taken from an invalid section or assigned to the wrong field. For example, an absolute maximum rating should not be used as a recommended operating condition, and supply voltage should not be compared with an analog signal output range. Therefore, the proposed method extends evidence-grounded question-answering with specification-type validation, unit checking, source-section control, and rule-based comparison for IoT device compatibility.

Table 1 summarizes the capability-level comparison between the related studies and the proposed method.

Table 1. Capability-level comparison between related studies and the proposed method.

Characteristic	[25]	[26]	[27]	[31]	[32]	[33]	[35]	[36]	[38]	[39]	[34]	[30]	[37]	Proposed
Retrieval-augmented question-answering	○	○	○	×	×	×	○	×	×	×	×	○	×	○
Technical document processing	×	×	×	×	○	○	×	×	×	○	○	○	×	○
Structured representation or information extraction	×	×	×	○	○	○	×	×	○	○	○	○	×	○
Tool-mediated reasoning	×	×	×	×	×	×	○	○	×	×	×	×	○	○
Source traceability	×	×	×	×	×	×	×	×	○	○	×	×	×	○
Specification-level datasheet schema	×	×	×	×	×	×	×	×	×	×	×	×	×	○
IoT compatibility reasoning	×	×	×	×	×	×	×	×	×	×	×	×	×	○

Here, ○ indicates that the capability is explicitly addressed by the study, whereas × indicates that the capability is not a central component.

Compared with these studies, the proposed method focuses on answer validation for IoT datasheet inquiry, where retrieval, structured fact extraction, source checking, and compatibility reasoning are integrated into an *agentic* RAG workflow.

3. Problem Definition and Motivation

In this section, we define the datasheet question-answering task and discuss the limitations of conventional top-*k* RAG.

3.1. Datasheet Question-Answering Task

This study focuses on question-answering over IoT device datasheets. For a practical IoT application system construction and deployment, a user often combines sensors, relays, communication modules, gateways, edge devices, and other embedded components. These devices may differ in supply voltage, signal type, communication interface, operating condition, and supported connection method. Therefore, interoperability among devices remains a central issue in the system design [40–42]. For non-expert users, the difficulty is not only finding a relevant specification, but also determining whether the found specification is the correct one for the intended setup.

The task of IoT datasheet question-answering is to generate an answer that is grounded in a datasheet evidence. Given a user question and one or more datasheets, the system should identify the relevant device, specification type, value, unit, operating condition, and supporting source when such information is available. For example, when a user asks about the operating voltage of a device, the system should return the recommended supply voltage rather than an unrelated voltage value from an absolute maximum rating, an example circuit, or a signal output description.

In this paper, each datasheet question-answering task is treated as an evidence-grounded technical question. The input is a natural-language question together with the datasheets of the target device or device pair. The output is an answer supported by the valid datasheet evidence. For a simple question, the answer may be a single specification value with its unit and source. For a structured question, the answer may contain multiple specifications. For a compatibility question, the answer should include a judgment, such as compatible, incompatible, or insufficient evidence, based on validated specifications and applicable comparison rules.

3.2. Question Types and Difficulty Levels

The benchmark questions are grouped into three difficulty levels: easy, medium, and hard. The *difficulty level* is defined according to the operation required to obtain the answer from datasheet evidence. *Easy questions* usually require a direct lookup of one explicit specification, such as supply voltage, communication interface, or output type. *Medium questions* require structured extraction of multiple specifications, condition-aware interpretation, or comparison between devices or variants. *Hard questions* involve compatibility reasoning, such as checking whether the output voltage range of a sensor fits the allowable input voltage range of a receiving module.

The benchmark is also categorized by the *question type*. *Simple question-answering* asks for one specification from one device. *Structured question-answering* asks to return several specifications in a structured format. *Comparison question-answering* asks for differences or similarities between devices or variants. *Compatibility question-answering* asks whether two devices can work together under a given technical condition. Table 2 summarizes the relationship between difficulty levels, representative question types, and required operations.

Table 2. Difficulty levels and question types in IoT datasheet question-answering.

Difficulty	Representative Type	Required Operation
Easy	Simple QA	Direct lookup of one specification
Medium	Structured QA/Comparison QA	Structured extraction or specification comparison
Hard	Compatibility QA	Unit normalization and rule-based compatibility judgment

This categorization is important because different question types require different reasoning processes. A conventional top-*k* retrieval pipeline treats these questions in a similar retrieval-and-generation manner. In contrast, the proposed method selects different processes according to the intent of the question, such as fact lookup, structured extraction, comparison, or compatibility checking.

For compatibility question-answering, the system must avoid confusion among different types of specifications. For example, a sensor supply voltage should not be directly compared with the analog input range of a receiving module. Similarly, an absolute maximum rating should not be treated as a recommended operating condition. Therefore, the difficulty of the task depends not only on the number of retrieved chunks, but also on whether the system can identify the correct specification type and apply the correct rule.

3.3. Limitations of Top- k RAG

A conventional top- k RAG system retrieves the most similar chunks from the document collection and then prompts an LLM to generate an answer. Modern neural retrieval methods can effectively rank passages by semantic similarity [43]. However, semantic similarity alone is not sufficient for IoT datasheets. A retrieved chunk may contain words that are relevant to the query, but the value in that chunk may not be the correct specification for the intended question. For example, several chunks sometimes contain the word “voltage”, but they can refer to supply voltage, signal output voltage, input voltage range, or absolute maximum rating.

Recent studies also suggest that retrieval is not always uniformly helpful and that language models may require external evidence selectively, depending on the factual content and task condition [44].

This limitation becomes more serious when several retrieved chunks are provided together. A prior work has shown that language models may not use long contexts uniformly and can miss or underuse relevant information depending on its position in the context [45]. In datasheet question-answering, this means that even if the correct chunk is retrieved, it may be mixed with other chunks that contain similar terms but describe different technical specifications. As a result, the generated answer may appear relevant in wording but still be wrong in technical meaning.

Another limitation is that top- k RAG does not provide explicit evidence control. It does not naturally check whether a value has a valid unit, whether the source section is appropriate, or whether the specification type is valid for a compatibility judgment. Therefore, the system can still generate an answer when the available evidence is incomplete or technically unsuitable. This behavior is problematic for IoT setup assistance, where an incorrect compatibility judgment may lead to wrong wiring, unstable operation, or device damage.

These limitations motivate the proposed method presented in the next section. Instead of treating datasheet question-answering as a direct retrieval-and-generation task, the proposed method treats it as specification-aware question-answering. It first identifies the required specification type, such as supply voltage, signal output voltage, input range, or communication interface, and then extracts the corresponding value, unit, condition, and source information from the datasheet. The extracted specification is validated before it is used in the final answer. For compatibility-related questions, the proposed method further applies explicit comparison rules over validated specifications, rather than relying only on the LLM to interpret top- k retrieved chunks.

4. Proposed AI Answer-Validation Method

Figure 2 shows the overall design and main execution flow of the proposed method.

The proposed method therefore uses the language model as a constrained process controller rather than as the sole technical decision maker. Given a user question, the agent selects an appropriate process, such as structured fact query, optional evidence retrieval, validation, or compatibility comparison. The final answer is accepted only when the required specification, value, unit, condition, and source information are verified. Otherwise, the method returns N/A or insufficient evidence.

The method consists of two main stages. In the *offline stage*, datasheets are parsed into context-preserving evidence chunks and structured facts, and extracted values are normalized before storage. In the *online stage*, the agent classifies the question type, queries structured facts, retrieves additional evidence when needed, and applies validation checks, including compatibility rules when required, before answer generation. This design moves

datasheet question-answering from direct retrieval-based generation to specification-aware and evidence-controlled answering.

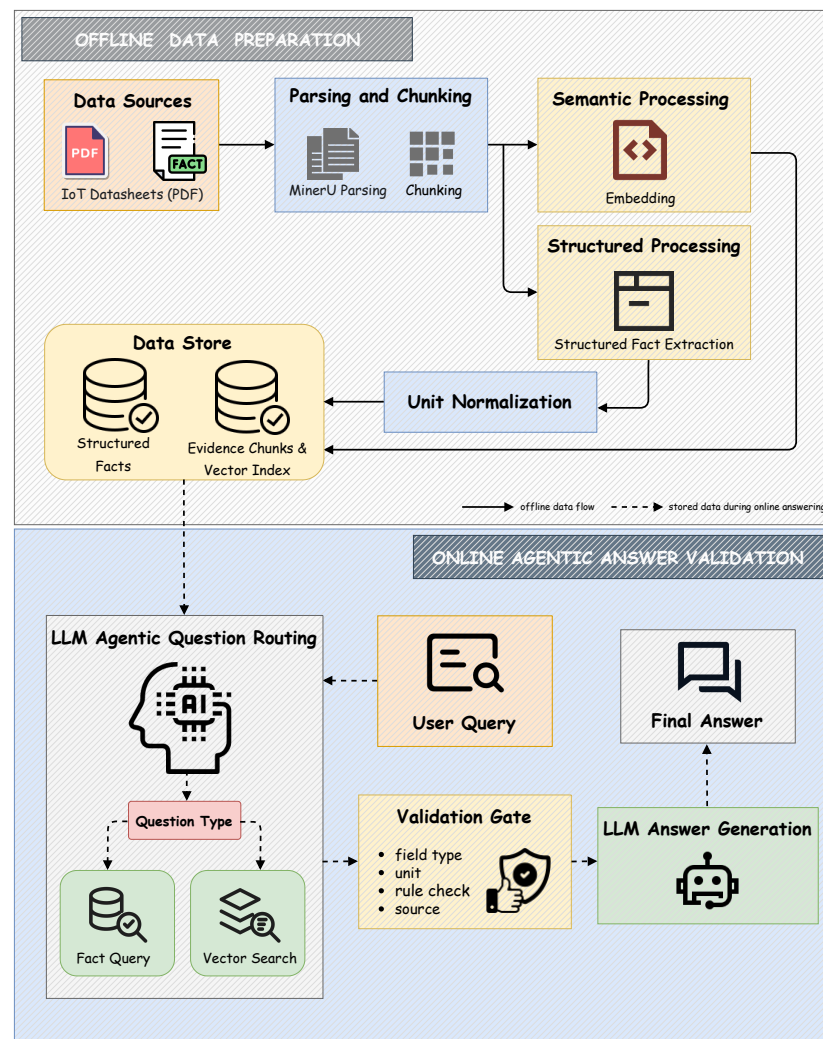


Figure 2. Overview of AI answer-validation method.

4.1. Offline Datasheet Representation

The *offline stage* prepares datasheet content for later question-answering. Its purpose is not only to make the documents searchable, but also to preserve the technical context needed to interpret specification values correctly. The output of this stage includes retrievable text chunks and structured specification facts linked to their source locations.

PDF datasheets are first converted into machine-readable text while retaining source information such as document name, page number, section title, and local text context when available. This metadata is necessary because datasheet values are often meaningful only within a specific section, table, or explanatory note. For example, a value reported in an electrical characteristics table should not be interpreted in the same way as a value reported in an absolute maximum rating section.

In the implementation, *MinerU* parsed each *PDF* page into Markdown evidence blocks. Each extracted block retained its document identifier and original page index. The preprocessing procedure grouped consecutive Markdown blocks into a chunk and stored the page index of the contributing blocks as source metadata. To preserve local context, each chunk also retains the nearest preceding Markdown heading when available, together with the immediately preceding and following text blocks. Therefore, a retrieved chunk

could be traced to its source datasheet page and interpreted together with its surrounding technical context.

Datasheet text is divided into context-preserving chunks so that the specification names, numerical values, units, and nearby conditions remain together as much as possible. This step is necessary because technical values may lose their meaning when separated from their table headers, row labels, footnotes, or operating conditions. A chunk that contains only a value such as “5V” is not sufficient for technical question-answering unless the system also knows whether the value refers to supply voltage, output voltage, input range, or another specification type.

Figure 3 illustrates the motivation for context-preserving segmentation. A direct text split may separate a specification name from its value, unit, or condition, producing ambiguous retrieved evidence. To reduce this risk, adjacent chunks retain a small shared context region. This overlap represents repeated local context between neighboring chunks rather than duplicated device information. As a result, technical values are more likely to remain associated with the specification names and descriptions required for reliable extraction.

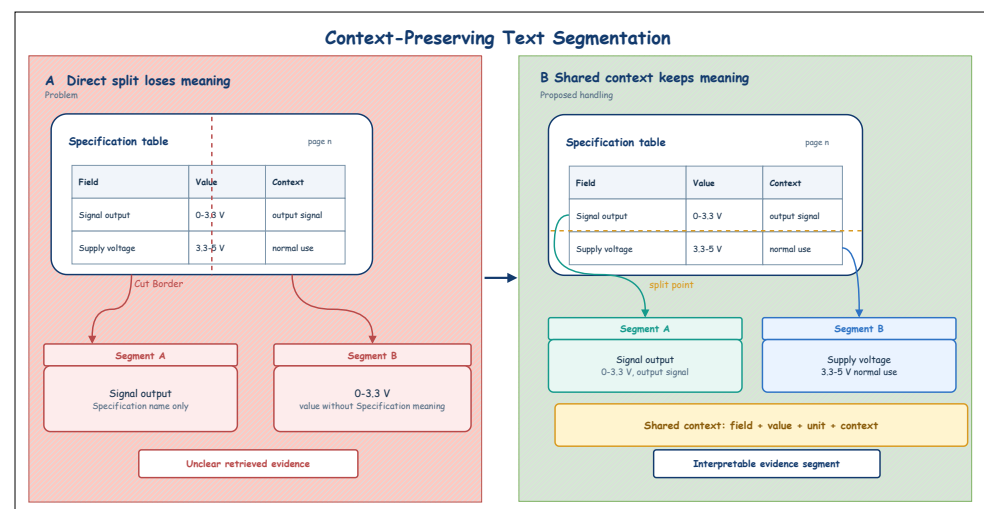


Figure 3. Context-keeping datasheet chunking strategy.

The resulting chunks are used by the semantic retrieval path. Each chunk is linked to its source document and page so that retrieved information can later be traced back to the original datasheet. When the structured fact memory does not contain the requested specification or when additional evidence is needed, the online agent can retrieve these chunks as supporting source text.

4.1.1. Schema-Guided Fact Representation

Structured specification facts are used as the intermediate representation between raw datasheet text and final answers. Each fact records one specification item together with its device model, specification name, value, unit, applicable condition, source section, source chunk, and validation status. This representation helps to reduce confusion among technically similar values before an answer is produced.

Figure 4 shows the conversion of unstructured datasheet content into schema-organized specification facts. The schema provides a fixed vocabulary and output structure for the extracted specifications. Instead of storing all technical values as plain text, the method stores values under separate specification types, such as supply voltage, signal output, input range, communication interface, operating condition, and supporting evidence.

Table 3 summarizes the predefined specification schema used for datasheet extraction.

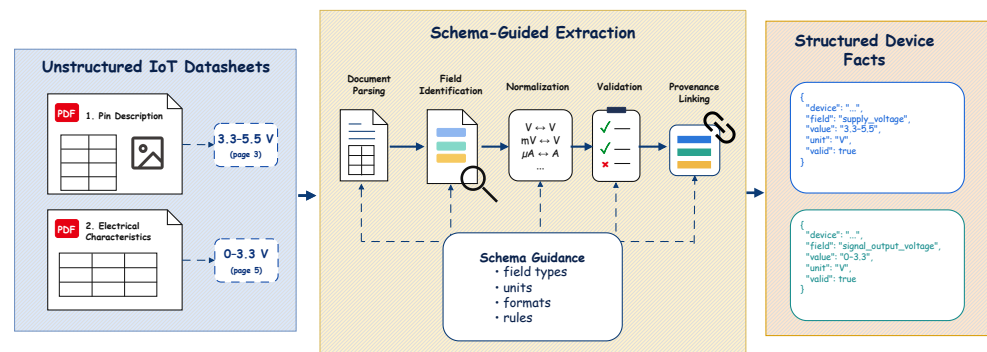


Figure 4. Schema-guided datasheet extraction.

Table 3. Specification schema for datasheet extraction.

Category	Extracted Specifications
Device identification	Model, manufacturer, function, order number, approvals
Input specifications	Sensor type, signal type, input range, isolation
Communication	Protocol, physical interface, communication parameters
Environmental and mechanical	Temperature, humidity, IP rating, mounting type
Setup and configuration	Supply voltage, power consumption, configuration, wiring, response time

The schema has two main functions. First, it limits the extraction to predefined specification items so that the system does not freely invent unsupported attributes. Second, it standardizes how extracted values are represented, including value, unit, condition, and source information. Missing values are explicitly represented as unavailable rather than inferred by the language model. This design helps the system to keep similar values under separate schema entries, rather than treating all voltage, current, or interface descriptions as interchangeable text.

4.1.2. Offline Unit Normalization

During offline fact construction, numerical specifications with recognized units are processed by a deterministic unit-normalization function before they are inserted into the `structured_facts` collection. The function programmatically parses a numerical value or range and its unit, maps units of the same physical dimension to a predefined base unit, and stores both the original representation and the normalized representation. For example, a value recorded as 3300 mV is preserved as the raw source value and additionally stored as 3.3 V for comparison. The supporting evidence text and source page remain unchanged. This conversion is completed during offline staging and is not delegated to the online language model.

During online question-answering, compatibility rules use normalized values only when the selected specifications satisfy a predefined field-type relation and have compatible physical dimensions. Thus, a signal output voltage may be compared with an input voltage range after unit normalization, whereas a supply voltage and a signal output voltage are not treated as comparable merely because both use volts. If a required unit is missing, cannot be parsed, or cannot be mapped to the required physical dimension, the method returns insufficient evidence rather than making a numerical comparison.

4.1.3. Persistent Fact and Evidence Storage

The JSON-based device facts are stored in *MongoDB* using two collections. The `structured_facts` collection stores normalized technical specifications, including the device identity, variant, field type, numerical value, unit, condition, evidence text, and page information. The `evidence_chunks` collection stores parsed document chunks and

their embedding vectors for semantic retrieval. Therefore, exact parameter queries can be executed as structured database lookups, while broader questions are handled through vector-based retrieval over evidence chunks.

During online inference, the agent selects the query path according to the user question. If the question requires an exact technical value, the system queries `structured_facts`. If the question requires broader contextual reasoning, the system retrieves relevant chunks from `evidence_chunks`. Compatibility-related questions first retrieve structured facts and then apply deterministic validation rules. This separation clarifies the storage and retrieval path used for each type of query.

4.2. Online Agentic Question-Answering

The *online stage* starts when a user submits a question. Instead of directly sending the question and retrieved chunks to the language model, the method first determines what type of operation is required. The agent then coordinates the structured fact lookup, text retrieval, source checking, and answer generation according to the question type.

In this paper, the term *agentic RAG* refers to a constrained workflow in which the language model selects the processing route and coordinates retrieval, structured specification access, and evidence checking. The agent is not used as an unrestricted autonomous decision maker. Technical acceptance is controlled by specification-type validation, source checking, and deterministic compatibility rules.

Algorithm 1 makes the control boundary of the online agent explicit. The *LLM* is limited to selecting one predefined question route and organizing an answer from validated evidence. It cannot independently accept a specification value or determine device compatibility. These decisions are controlled by the fixed field, unit, condition, source, and compatibility checks.

Algorithm 1 Constrained answer-validation workflow.

Require: Question q , fact registry F , Markdown chunks C , policies V , rules R

Ensure: Supported answer a , N/A, or insufficient evidence

```

1:  $t \leftarrow \text{ROUTEQUESTION}(q); S \leftarrow \text{REQUIREDSPECIFICATIONS}(q, t)$ 
2:  $B \leftarrow \text{GETFACTEVIDENCE}(F, S)$ 
3: if  $\text{NEEDSTEXTEVIDENCE}(B, q)$  then
4:    $B \leftarrow \text{ADDTOPKCHUNKS}(B, C, q)$ 
5: end if
6: if  $\text{VALIDATEEVIDENCE}(B, q, V) = \text{false}$  then
7:   return N/A or insufficient evidence
8: end if
9: if  $t = \text{compatibility}$  then
10:   $a \leftarrow \text{APPLYCOMPATIBILITYRULES}(B, R)$ 
11: else
12:   $a \leftarrow \text{FORMATEVIDENCEBOUNDANSWER}(q, B)$ 
13: end if
14: if  $\text{ANSWERSUPPORTEDBYEVIDENCE}(a, B) = \text{false}$  then
15:  return insufficient evidence
16: end if
17: return  $a$ 

```

4.2.1. Question-Type Routing

The question router classifies user questions into different processing routes. Simple factual questions are handled by single-specification lookup. Structured questions require multiple specifications to be collected and returned in a consistent format. Compatibility questions require the pairwise evidence extraction from two devices before rule-based comparison can be applied.

This routing step is important because different question types require different evidence and reasoning processes. A conventional top- k RAG pipeline treats these questions in a similar retrieval-and-generation manner. In contrast, the proposed method uses the question type to decide whether the answer should be produced by direct lookup, structured extraction, source checking, or compatibility reasoning.

For factual questions, the agent first queries the structured specification facts. If the requested information is unavailable or insufficiently supported, the agent retrieves relevant text chunks as additional evidence. For structured questions, the agent collects multiple schema entries and returns them in a unified format. For compatibility questions, the agent obtains the relevant specifications of both devices before passing them to the compatibility reasoning module.

4.2.2. Source Checking and Abstention

Before an answer is accepted, the proposed method checks whether the selected specification is supported by a valid datasheet source. The answer must be linked to the source content that contains the reported value and its technical meaning. For factual and structured questions, the returned value must be supported by either the structured specification fact or the retrieved text chunk. For compatibility questions, each compared specification must be supported before the final judgment is generated.

If the required value, unit, condition, or source information is missing, the method abstains by returning N/A or insufficient evidence instead of generating a speculative answer. This behavior is important for technical question-answering where an unsupported answer may be more harmful than no answer. The abstention mechanism therefore treats missing evidence as a controlled outcome rather than as a generation failure.

This source-checking design also reduces the risk of accepting technically plausible but unsupported responses. The language model can organize candidate answers and supporting sources, where the answer is not accepted solely because it is fluent or semantically related to the question. It must pass the predefined source checks before being returned to the user.

4.3. Compatibility Reasoning

Compatibility reasoning is applied when the question requires cross-device comparison. The method first extracts the relevant specifications from each device and checks whether the two specifications are technically comparable. A compatibility judgment is produced only after the required values, units, conditions, and source information have passed validation.

For analog-voltage compatibility, the analog output range of the source device is compared with the input range of the receiving device. This rule reflects the engineering requirement that the signal produced by one device should fall within the acceptable input range of the next device. For example, a sensor output range can be compared with an analog-digital converter input range, because both describe signal-level compatibility across a connection path.

In contrast, supply voltage is not directly compared with signal voltage. Supply voltage describes the power requirement of a device, whereas signal voltage describes the electrical level carried by an output or input signal. Comparing these two values directly may produce a technically invalid compatibility judgment, even if both values are expressed in volts. Therefore, the method requires the specification types to be compatible before a numerical range comparison is performed.

For communication compatibility, the method checks whether the participating devices report compatible interface types, such as *I2C*, *SPI*, *UART*, or other supported protocols. When available, related information such as pin assignment, bus address, voltage level, and communication mode can also be used as supporting evidence. If the interface information

is missing or if the two devices report incompatible protocols, the system does not force a compatible judgment.

When the datasheet does not provide enough evidence for a required comparison, the method returns the insufficient evidence. This rule applies when one of the compared values is missing, when the unit is unavailable, when the source section is not suitable, or when the two specifications are not technically comparable. As a result, compatibility reasoning is constrained by explicit evidence and engineering rules rather than by unrestricted language-model generation.

This rule-based design makes the compatibility decision more controllable than direct top-*k* RAG generation. The language model coordinates evidence selection and answer organization, while the technical comparison is restricted by predefined compatibility rules. This separation helps to prevent invalid cross-specification comparisons, such as comparing a sensor supply voltage with an analog-digital converter input range or treating an absolute maximum rating as a recommended operating condition.

5. Implementation of Proposed Method

In this section, we describe the implementation of the proposed method. While the previous section explains the design of the workflow, this section focuses on how datasheets are parsed, chunked, indexed, queried, and checked in the implemented system.

5.1. Local PDF Parsing

The implementation begins with local processing of IoT datasheets. A locally deployed parser converts each *PDF* document into machine-readable text without sending the original document to an external parsing service. The parser keeps page numbers, section headings, table content, technical notes, and local layout information when they can be reliably identified. These elements are retained because important device specifications are often distributed across electrical characteristic tables, pin descriptions, recommended operating conditions, and explanatory notes.

The extracted content is stored with source metadata, including the originating document and page number. Section information and local text positions are also retained when available. Repeated headers, footers, isolated page numbers, and unrelated copyright statements are removed, whereas field labels, units, table headings, and footnotes are preserved. This processing reduces irrelevant text while keeping the local context required for later source checking.

5.2. Chunk Construction

Our implementation follows the context-keeping chunking strategy described in Section 4. After datasheet processing, the cleaned text is organized in the document order and divided into controlled-length chunks. A small overlap is retained between the neighboring chunks so that boundary cases do not remove the local context needed to interpret a technical value.

In the implementation, the chunk construction is applied after page-level metadata has been assigned. Each chunk therefore keeps its source document, page number, and local text content. When table-like content is detected, the system attempts to keep table headers, row labels, values, and nearby notes in the same retrieval unit. This treatment is especially important for voltage-related fields, where similar numerical ranges may represent supply voltage, signal output voltage, input voltage range, or absolute maximum rating.

It is noted that this subsection concerns how the chunking policy is operationalized. The chunks are not treated as final answers; they are stored as source candidates for semantic retrieval, field extraction, and post-generation source checking.

5.3. Schema and Memory

The implementation uses a predefined specification schema to constrain the structured output. The schema specifies the target categories and fields used in the current system, including the device identification, input specifications, communication, environmental and mechanical specifications, and setup-related specifications. The schema is used to generate a consistent extraction format, so that the same type of specification is represented in the same field across different datasheets.

Extracted specifications are stored as *JSON*-based device facts. Each stored fact records the device model, specification name, reported value, unit when available, applicable condition, and supporting source information. Missing values are represented explicitly instead of being inferred by the language model. If a generated structured output cannot be parsed as valid *JSON*, it is rejected and not used as reliable fact memory.

The fact memory is used as a reusable structured store. When a later question requires the same specification, the system first checks the stored device fact. If the required field is absent, inconsistent, or insufficiently supported, the system returns to the source chunks for additional retrieval rather than relying on an unsupported stored value. This avoids repeated long-context prompting for already available specifications while still allowing the system to consult the original chunks when the stored data is incomplete.

Listing 1 shows the prompt template used for the specification extraction. The template constrains the language model to extract only explicitly supported specifications and to return a structured *JSON* object with source evidence.

Listing 1: Prompt template for specification extraction.

```

### Instruction ###
You are an IoT datasheet extraction assistant.
Extract only the requested technical specifications from the provided datasheet
context.
Do not infer missing values.

### Target Specifications ###
{{TARGET_SPECIFICATIONS}}

### Datasheet Context ###
{{CONTEXT}}

### Output Format ###
Return a JSON object with:
- device
- specification_name
- value
- unit
- condition
- source_section
- evidence_text

If a value is not explicitly supported, return null.
```

This prompt is used only for converting retrieved datasheet context into structured device facts. Compatibility judgment and answer acceptance are handled by validation rules and evidence checking rather than by unrestricted prompt generation.

5.4. Retrieval and Tools

The retrieval component uses the target device, requested specification, and question content to retrieve relevant source text. This is useful for datasheets because device models, interface names, units, and pin identifiers provide important cues for locating the relevant technical information.

Qwen-Plus is used as the language model for question interpretation and controlled response generation. In the proposed workflow, the model does not directly determine the final technical judgment from raw retrieved text. Instead, it identifies the requested information, organizes the required specifications, and coordinates retrieval, extraction, and checking steps. For factual questions, the method queries the corresponding specification. For structured questions, multiple specifications are collected according to the schema. For compatibility questions, the relevant specifications of both devices are obtained before rule-based comparison is applied.

The tool layer provides an execution interface between the language model and the information sources. It supports source retrieval, structured specification access, missing-value handling, and source-aware response formatting. This design keeps the model's role focused on process selection and source organization, while technical acceptance is handled by constrained validation and rule checking.

5.5. Validation and Rules

Before an extracted value is used, the method checks whether the value has a compatible field type, unit, condition, and supporting source. Values from sections with different technical purposes are not treated as interchangeable. For example, an absolute maximum rating is not used as a recommended operating condition, and a storage condition is not used as a runtime specification.

Compatibility checking is implemented only over validated and technically comparable fields. For analog-voltage compatibility, the signal output range of the source device is compared with the input range of the receiving device. Supply voltage is not compared directly with signal output voltage because the two fields describe different electrical properties. Communication compatibility is checked using the reported interface information of the participating devices.

If a required value, unit, condition, or supporting source is missing, the system returns insufficient evidence rather than producing a speculative judgment. The generated answer is also checked against the associated source string. Unsupported values are rejected and represented as N/A. This implementation prevents technically inappropriate retrieved content from being accepted as a final answer.

5.6. Top-k Baseline

The baseline in our previous study was implemented as a conventional top-*k* RAG pipeline. For each question, the highest-ranked document chunks are retrieved and concatenated as context for *Qwen-Plus*, which then generates the answer directly. The baseline does not use question-dependent process selection, structured fact memory, schema-based validation, compatibility rules, or post-generation source checking.

The baseline and the proposed method use the same datasheet collection, local document processing results, and language model. Using the same datasheet collection, local document processing results, and language model isolates the effect of structured facts, source checking, and rule-based compatibility reasoning from the effect of document parsing or model choice.

6. Evaluation

In this section, we present the experimental setup and results for evaluation.

6.1. Benchmark Dataset Construction

For evaluation, a benchmark dataset was constructed from 20 IoT datasheets. The datasheets cover common IoT-related devices, including sensors, regulators, analog-to-digital converters (ADCs), relays, current sensors, process sensors, and other components that require users to inspect technical specifications before use. These devices were selected because their configuration information is often distributed across different datasheet sections, such as electrical characteristics, pin descriptions, recommended operating conditions, interface descriptions, and application notes.

The selection aimed to include both sensing devices and supporting components with different signal types, communication interfaces, and electrical constraints. The selected datasheets cover temperature and humidity, ambient light, gas and air quality, pressure, pH/ORP, water flow, current, and motion-related sensing, together with regulators, analog-to-digital converters, relays, motor drivers, and signal-conditioning components. This range supports factual lookup, multi-item extraction, and cross-device compatibility questions.

From these datasheets, 1000 question-answering tasks were prepared. The questions were manually written to reflect common datasheet-based setup tasks. The questions were checked to ensure that the reference answers could be traced to datasheet evidence. This treatment was intended to reduce unsupported labels and to make the benchmark closer to practical datasheet consultation. The questions were divided into three difficulty levels according to the type of evidence and reasoning required. *Easy questions* require a single-device and single-specification factual lookup. *Medium questions* require multiple specification items, structured outputs, variant-specific extraction, or condition-specific extraction. *Hard questions* require cross-device comparison, compatibility-related reasoning, or careful distinction between technically similar specifications.

Table 4 summarizes the composition of the IoT datasheet Q&A benchmark by difficulty level.

Table 4. Summary of IoT datasheet Q&A benchmark.

Part	Number	Task Focus
Datasheets	20	Common IoT devices and components
Easy QA	500	Single-device factual lookup
Medium QA	350	Structured or condition-specific extraction
Hard QA	150	Cross-device comparison and compatibility reasoning
Total QA	1000	Three difficulty levels

Each question was paired with a reference answer prepared and manually verified by the authors, who have research backgrounds in IoT application systems; no independent blinded experts were involved. The annotation guideline required each reference answer to be explicitly supported by datasheet evidence. For numerical questions, the device model or variant, specification type, value, unit, applicable condition, and source location were checked. For compatibility-related questions, the relevant specifications of both devices were checked before applying the corresponding comparison rule. Questions whose reference answers could not be clearly supported by the datasheet were excluded or revised during benchmark construction.

Table 5 presents representative question templates for the three difficulty levels.

Table 5. Representative question templates by difficulty level.

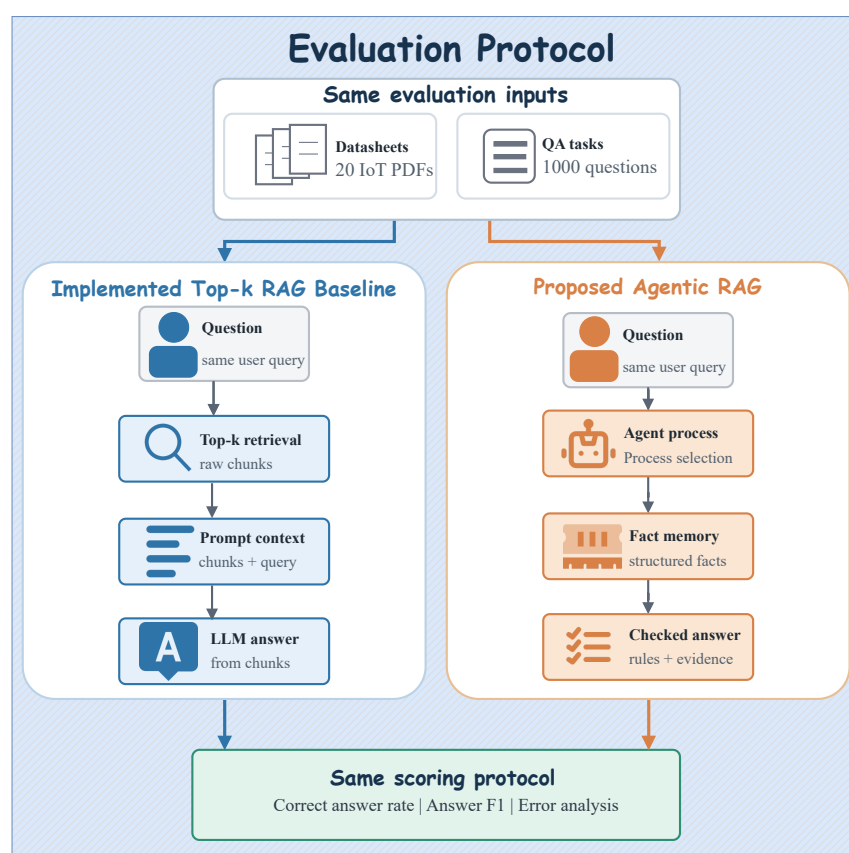
Difficulty	Question Template	Required Operation
Easy	What is the supply voltage of the device?	Single-specification lookup
Medium	Return the supply voltage and communication interface as separate items.	Structured extraction
Hard	Is the sensor output voltage compatible with the ADC input range?	Compatibility reasoning

6.2. Compared Methods and Baselines

The evaluation compares the top- k RAG baseline and the proposed method under the same datasheet collection, question set, language model, and scoring protocol. For each question, this baseline retrieves the top-ranked chunks from the datasheet index and asks the language model to generate an answer based on the retrieved context. In this setting, the retrieved text is directly used as the main evidence source for answer generation.

The proposed AI answer-validation method uses *agentic* RAG. Unlike the baseline, the proposed method first determines the question type and then selects the corresponding process, such as fact lookup, structured extraction, evidence checking, or compatibility-related comparison. It uses structured device facts, specification validation, evidence verification, and rule-based comparison before accepting the final answer.

Figure 5 summarizes the evaluation protocol. The same benchmark questions are given to both methods. Their generated answers are then compared with the manually verified reference answers using the same scoring protocol. This evaluation design ensures that the comparison focuses on the effect of agentic process selection, structured fact usage, evidence control, and rule-based reasoning rather than differences in the dataset or evaluation criteria.

**Figure 5.** Evaluation protocol for method comparison.

6.3. Evaluation Metrics

The evaluation uses answer-level metrics and error analysis. The primary metric is the correct answer rate, because the benchmark contains manually verified reference answers for all questions. An answer is counted as correct only when it matches the expected technical meaning in the reference answer. For numerical specifications, the specification type, value, unit, and applicable condition must be consistent with the datasheet evidence. For compatibility questions, the final judgment must be derived from the correct pair of technical specifications.

The correct answer rate is computed as

$$\text{Correct Answer Rate} = \frac{N_{\text{correct}}}{N_{\text{total}}}$$

where N_{correct} denotes the number of correctly answered questions and N_{total} denotes the total number of evaluated questions.

For questions that require one or more answer items, the answer F1 is used to evaluate partial correctness. An answer item may be a technical specification, a unit, an applicable condition, or a compatibility judgment, depending on the question type. Precision and Recall are computed over the returned and required items, respectively, and the F1 score is defined as

$$\text{F1} = \frac{2 \times \text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}$$

Evidence support is considered during manual answer verification rather than reported as a separate ranking metric. This choice follows the practical nature of the task: an answer is accepted only when the reported specification and its technical meaning are supported by the datasheet. Inspired by evidence-aware RAG evaluation such as RAGAs [46], the verification checks whether the cited datasheet content supports the answer, whether the relevant specification type is used, and whether the unit or condition is consistent with the source document.

Error analysis is conducted to identify the main failure causes of each method. The error categories are defined as E1, wrong specification selection; E2, incomplete output; E3, missing answer; E4, unit or format error; and E5, missing evidence. These categories are used to explain the remaining errors after the answer-level evaluation.

6.4. Experimental Protocol

The experiments were conducted on the same 1000-question benchmark for both the baseline and the proposed method. The questions were grouped into easy, medium, and hard levels before evaluation, and the results were reported both overall and by difficulty level. This grouping allows the evaluation to show whether the proposed method mainly improves simple fact lookup or whether it also benefits more difficult compatibility and comparison questions.

The results for the three difficulty groups are reported separately in Figures 6 and 7. Therefore, the performance reported for the hard questions is calculated only from the hard-question subset and is not affected by the larger number of easy questions.

The local workstation was used for document parsing, indexing, retrieval, and result analysis, while *Qwen-Plus* was used for language-model inference. The same processed datasheets, retrieval index, language model, and scoring protocol were used for the implemented top- k RAG baseline and the proposed method. This setting ensures that the comparison focuses on the effect of structured specification data, specification validation, source checking, and rule-based compatibility reasoning rather than differences in the

document collection or language model. The main implementation settings are summarized in Table 6.

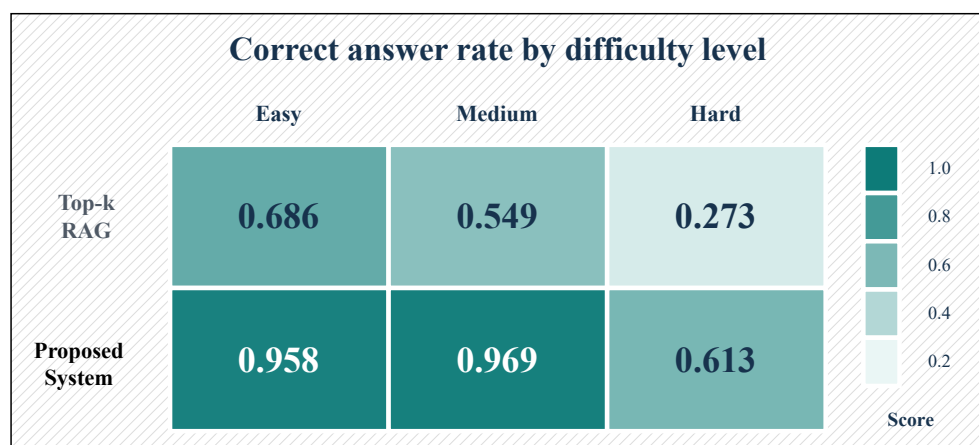


Figure 6. Correct answer rate by difficulty level.

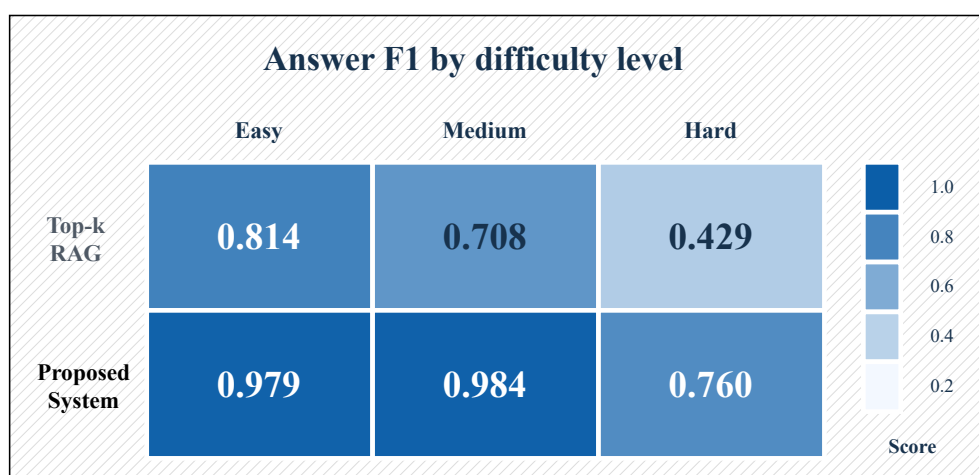


Figure 7. Answer F1 by difficulty level.

Table 6. Main implementation settings.

Item	Setting
Parser	MinerU
Chunking	1200 chars, 180 overlap
Embedding	MiniLM-L12-v2
Retrieval	Cosine Top- k , $k = 10$
Storage	MongoDB
LLM	Qwen-Plus
Generation	temperature 0.0, 128 tokens
Environment	i7-12700F, RTX 5070 Ti, 32 GB RAM, Windows 10

For the top- k RAG baseline, the retrieved chunks are directly concatenated as context for the language model. The model then generates the answer without structured fact memory, specification validation, compatibility rules, or post-generation evidence checking. Therefore, the baseline tests whether standard retrieval and generation are sufficient for datasheet question-answering.

For the proposed method, the agent first identifies the question type and selects the corresponding process. *Easy questions* are handled mainly through fact lookup. *Medium questions* are handled through multi-specification extraction and structured fact checking. *Hard*

questions are handled through pairwise specification extraction, specification validation, and rule-based comparison. If the required evidence is missing or technically insufficient, the proposed method is allowed to return N/A rather than producing an unsupported answer.

The final evaluation compares the two methods using the same manually verified reference answers. Error cases are further analyzed according to their causes, including incorrect retrieval, specification confusion, missing unit or condition, unsupported generation, and incorrect compatibility judgment. This analysis is used to explain not only whether the proposed method improves accuracy, but also why the improvement occurs across different question types.

6.5. Overall Performance Result

Figure 6 shows the correct answer rate of the implemented top-*k* RAG baseline and the proposed AI answer-validation method using *agentic* RAG across the three difficulty levels. The correct answer rate is used as the primary benchmark metric because each question has a manually verified reference answer.

The proposed method improves the correct answer rate from 0.686 to 0.958 for *easy questions*, from 0.549 to 0.969 for *medium questions*, and from 0.273 to 0.613 for *hard questions*. The largest absolute improvement appears in medium questions, where the proposed method benefits from structured specification extraction and specification validation. The hard-question score is still lower than the easy and medium scores because compatibility questions require evidence from multiple devices and correct specification-type comparison.

Figure 7 reports the answer F1 score by difficulty level. This metric is useful for questions that require one or more answer items because it reflects partial correctness when the returned answer is not completely correct.

The answer F1 score also improves across all difficulty levels. For *easy questions*, the score increases from 0.814 to 0.979. For *medium questions*, it increases from 0.708 to 0.984. For *hard questions*, it increases from 0.429 to 0.760. These results show that the proposed method improves not only the exact answer correctness but also the specification-item completeness in structured and comparison-oriented questions.

Table 7 reports the overall Precision and Recall. The proposed method returns fewer incorrect answer items and includes more required answer items than the top-*k* RAG baseline.

Table 7. Overall answer-item Precision and Recall.

Method	Precision	Recall
Top- <i>k</i> RAG	0.684	0.785
Proposed Method	0.935	0.971

6.6. Performance by Difficulty Level

The improvement pattern differs by difficulty level. *Easy questions* mainly require one factual specification, so the main benefit comes from direct specification lookup and evidence checking. In these cases, the proposed method reduces the risk that the language model generates an answer from a related but technically incorrect chunk.

Medium questions require multiple specification items, variant-aware values, or condition-specific values. The improvement in this group is the largest in correct answer rate. This result indicates that structured specification extraction is effective when the answer must contain several separate technical specifications, such as supply voltage and communication interface, or when values must be separated according to their conditions.

Hard questions remain the most challenging. These questions often require the system to collect facts from more than one device and compare technically compatible specifications. For example, a signal output voltage should be compared with an input voltage range,

whereas a supply voltage should not be treated as the same type of specification. Even in this setting, the proposed method gives a clear improvement over top- k RAG because it separates specification extraction, validation, and rule-based comparison before producing the final answer.

6.7. Ablation Study

To examine the contribution of the main components, an ablation study was conducted under the same dataset and scoring protocol. Table 8 reports the implemented top- k RAG baseline, two reduced variants of the proposed method, and the full proposed method. *Hard-QA* F1 denotes the answer F1 computed only on hard questions, which mainly involve cross-device comparison and compatibility-related reasoning.

Table 8. Ablation results on IoT datasheet Q&A benchmark.

Method	Disabled Module	Accuracy	F1	Hard-QA F1
Top- k RAG	All proposed modules	0.576	0.731	0.429
Proposed w/o structured validation	structured validation	0.303	0.465	0.370
Proposed w/o Rule Reasoning	Rule reasoning	0.847	0.917	0.462
Full Proposed Method	None	0.910	0.953	0.760

6.7.1. Effect of Structured Validation

This ablation isolates the contribution of answer validation in the extracted-fact answering process. In the “Proposed w/o Structured Validation” setting, the *LLM* agent still routes the question, queries the same extracted facts, and generates the answer. However, the selected facts are used without checking their field type, unit, operating condition, or supporting source.

This variant differs from the top- k RAG baseline, in which the *LLM* generates an answer from retrieved original datasheet chunks that retain field labels and local context. When a datasheet contains several related values, the ablation setting can use a plausible but incorrect fact, such as supply voltage in place of signal output voltage, whereas the validation process rejects this mismatch in the proposed method. Therefore, its lower performance is not a logical contradiction with the top- k RAG baseline; it shows that extracted facts require answer validation before they can be used as technical evidence.

6.7.2. Effect of Rule-Based Compatibility Reasoning

The variant without rule-based compatibility reasoning keeps a relatively high overall score because many easy and medium questions can still be handled by retrieval and structured extraction. However, its *Hard-QA* F1 remains much lower than that of the full method. This result suggests that rule-based compatibility reasoning mainly contributes to hard questions, where the proposed method must compare technically comparable specifications across devices.

The full proposed method achieves the best overall accuracy, overall F1, and *Hard-QA* F1. These results indicate that specification-type validation and rule-based compatibility reasoning contribute to different parts of the task: specification-type validation helps prevent wrong specification selection, while rule reasoning is especially important for compatibility-related questions.

6.8. Analysis by Question Type

The results can also be interpreted by question type. *Simple Q&A* benefits from direct specification lookup because the system extracts a specific technical specification instead of generating an answer from several related chunks. *Structured Q&A* benefits from structured

specification extraction because multiple specification items can be normalized into a consistent representation. *Compatibility Q&A* benefits from pairwise specification planning and rule-based reasoning because the system compares validated specifications instead of relying only on the *LLM*'s general interpretation.

These observations are consistent with the design motivation of the proposed method. The proposed method does not treat all questions as the same retrieval-generation task. Instead, it selects different processes for factual lookup, structured extraction, and compatibility-related reasoning. This process selection is particularly useful for medium and hard questions, where the answer depends on distinguishing technically similar specifications rather than merely retrieving semantically related text.

6.9. Error Analysis

Figure 8 summarizes the main error patterns observed in the evaluation. The purpose of this analysis is to explain where the two methods fail, rather than to introduce another performance metric. The error categories in Figure 8b are listed from E1 to E5 from top to bottom.

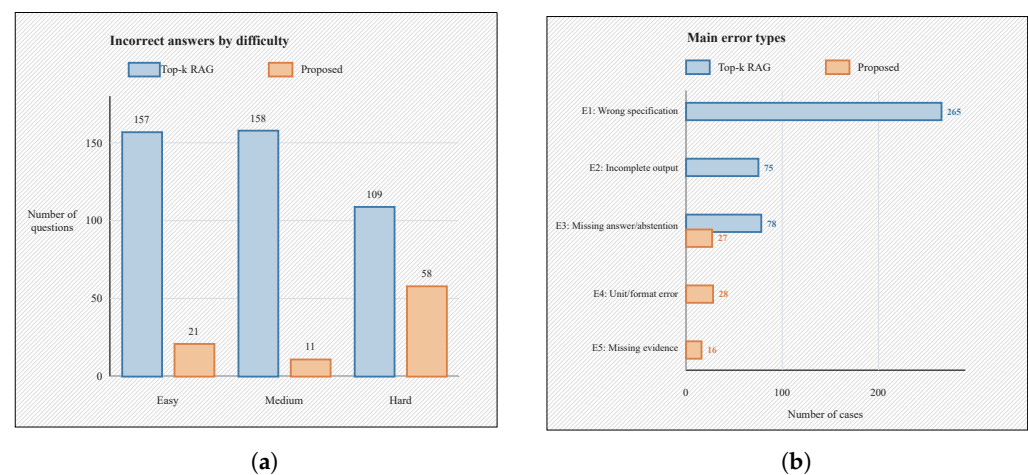


Figure 8. Error analysis by difficulty level and type: (a) number of incorrect answers by difficulty level; (b) main error types observed for the top-*k* RAG baseline and the proposed method.

Figure 8a shows that the proposed method reduces the number of incorrect answers across all difficulty levels. The reduction is especially clear for easy and medium questions, where structured specification extraction and evidence checking help distinguish similar datasheet values. For hard questions, more errors remain because cross-device compatibility requires evidence from multiple devices and comparison between technically comparable specifications.

Figure 8b further shows the distribution of the main error types. E1 denotes wrong specification selection, where the answer uses a related but technically incorrect specification. E2 denotes incomplete output, where only part of the expected answer is returned. E3 denotes a missing answer or abstention. E4 denotes unit or format errors. E5 denotes missing evidence, where the required supporting datasheet information is not retrieved or linked.

Figure 8a counts incorrect answers, whereas Figure 8b reports the main observed error patterns for the two methods. The error patterns in Figure 8b are method-specific and may co-occur within an incorrect answer. In addition, some incorrect answers may not belong to the five main patterns shown in Figure 8. Therefore, the bars in Figure 8 should be read separately for the blue top-*k* RAG baseline and the orange proposed method, rather

than added across methods or interpreted as a one-to-one partition of the incorrect-answer counts in Figure 8a.

For the implemented top- k RAG baseline, the dominant error type is E1. This indicates that the retrieved chunks may contain relevant technical text, but the final answer can still select the wrong specification when several similar values appear in the same context. This does not imply that RAG is inherently ineffective; it shows the limitation of answering datasheet questions directly from raw retrieved passages under the implemented baseline setting.

For the proposed method, the remaining errors are more often related to E3, E4, and E5. These errors indicate that the proposed method is more conservative: when the required specification, unit, or evidence cannot be verified, it is more likely to return an incomplete answer or abstain rather than generate an unsupported answer. This result supports the architectural motivation of using structured fact memory, evidence checking, and rule-based comparison to reduce repeated interpretation of raw datasheet text during online answering.

7. Discussion

In this section, we discuss the findings and limitations of the proposed method.

7.1. Effectiveness

The experimental results indicate that the proposed AI answer-validation method using *agentic* RAG is effective on the constructed IoT datasheet question-answering benchmark, because the task requires more than retrieving semantically related text. Datasheet questions often involve specification-level distinctions and engineering constraints. A retrieved chunk may contain relevant terms, but the answer can still be incorrect if the system selects the wrong specification type, ignores the applicable condition, or compares technically different values.

The quantitative results support this interpretation. The proposed method improves the correct answer rate by +0.272 for *easy questions*, +0.420 for *medium questions*, and +0.340 for *hard questions*. It also improves the answer F1 by +0.165, +0.276, and +0.331 for easy, medium, and hard questions, respectively. The improvement is most direct for easy and medium questions, where structured specification extraction and specification validation reduce confusion among similar datasheet values. For hard questions, the improvement is smaller but still important because compatibility reasoning requires information from multiple devices and comparison between technically comparable specifications.

From the user-support perspective, the proposed method reduces the need for non-expert users to manually locate electrical tables, distinguish similar voltage specifications, and decide whether two values are technically comparable. Therefore, the benefit is not only a higher answer accuracy, but also a simpler and more controlled procedure for checking device specifications and compatibility.

7.2. Reliability

Structured device facts can improve answer control by converting raw datasheet text into a normalized representation. Once a specification item is extracted, the proposed method records its specification name, value, unit, condition, source section, and evidence chunk. This representation makes it easier to verify whether the answer is supported by valid evidence and whether the extracted value is being used for the correct technical purpose.

Evidence checking also reduces unsupported generation. If the system cannot find valid evidence, it returns insufficient evidence instead of forcing an answer. This behavior

is important for the technical question-answering because an unsupported answer may be more harmful than no answer. The error analysis also supports this point: the remaining errors of the proposed method are more often related to missing answers, unit or format issues, and missing evidence, rather than selecting a wrong specification from a relevant but ambiguous chunk.

7.3. Limitations

The proposed method has several limitations. First, the quality of the answer depends on the quality of datasheet preprocessing. If tables, section titles, or local notes are not extracted correctly, specification extraction may fail. Second, some datasheets contain incomplete or ambiguous specifications. In such cases, the method may return insufficient evidence even when a human expert could make an informed assumption. Third, the current method focuses on specification items that are important for IoT datasheet question-answering, such as voltage, communication interface, input range, output type, and operating condition. Additional specification categories may be needed for broader device families.

Future work will extend the specification schema and compatibility rules to cover more device types and more complex setup scenarios. Another direction is to improve the handling of tables and figures, because some datasheet specifications are presented visually rather than in plain text. The evaluation can also be expanded with more datasheets and more diverse question types.

For a new datasheet, document parsing, chunking, fact extraction, unit normalization, embedding generation, and storage in *MongoDB* are completed in the offline stage and are not repeated for later questions about the same document. During online answering, the *LLM* agent selects the question path, queries the required structured facts, retrieves top-*k* evidence chunks only when additional text is required, and applies the relevant validation or compatibility rule. Thus, compared with direct top-*k* RAG, the additional online work consists of question routing, structured fact lookup, and checking of the selected facts. The present study evaluates answer quality on 20 datasheets and does not provide a production-scale throughput or end-to-end latency benchmark; evaluating these aspects on a larger datasheet collection is left for future work.

8. Conclusions

This paper proposed an AI answer-validation method using *agentic* RAG for datasheet inquiry in IoT application system deployment. The method addresses several limitations observed in the implemented top-*k* RAG baseline by organizing datasheet information into structured device facts and selecting different processes according to the question type. Instead of relying only on the *LLM* to interpret retrieved chunks, the proposed method uses specification validation, unit normalization, evidence checking, and rule-based compatibility reasoning.

The evaluation was conducted on a dataset constructed from 20 IoT datasheets, including 1000 question-answering tasks with three difficulty levels. The dataset contains 500 easy questions, 350 medium questions, and 150 hard questions. Compared with the implemented top-*k* RAG baseline, the proposed method improved the correct answer rate from 0.686 to 0.958 for easy questions, from 0.549 to 0.969 for medium questions, and from 0.273 to 0.613 for hard questions. The answer F1 also improved from 0.814 to 0.979 for easy questions, from 0.708 to 0.984 for medium questions, and from 0.429 to 0.760 for hard questions.

These results show that structured device facts and evidence-controlled reasoning can improve IoT datasheet question-answering in the constructed benchmark. The pro-

posed method is especially useful for questions that require specification distinction or compatibility judgment. By requiring answers to be supported by valid evidence and by separating specification extraction from rule-based comparison, the proposed method can reduce the risk of technically plausible but unsupported responses. Future work will extend the schema and rules to more device categories and improve the processing of complex datasheet layouts.

Author Contributions: Conceptualization, D.K. and N.F.; methodology, D.K. and N.F.; software, D.K., H.H.S.K., I.N.D.K. and Z.Z.; validation, D.K., N.F. and Z.Z.; investigation, D.K. and I.N.D.K.; resources, I.N.D.K.; writing—original draft, D.K.; writing—review and editing, N.F.; supervision, N.F. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: Data are contained within the article.

Acknowledgments: The authors thank the reviewers for their thorough reading and helpful comments.

Conflicts of Interest: The authors declare no conflicts of interest.

References

- Atzori, L.; Iera, A.; Morabito, G. The Internet of Things: A Survey. *Comput. Netw.* **2010**, *54*, 2787–2805. [\[CrossRef\]](#)
- Kortuem, G.; Kawsar, F.; Fitton, D.; Sundramoorthy, V. Smart Objects as Building Blocks for the Internet of Things. *IEEE Internet Comput.* **2010**, *14*, 44–51. [\[CrossRef\]](#)
- Gubbi, J.; Buyya, R.; Marusic, S.; Palaniswami, M. Internet of Things (IoT): A Vision, Architectural Elements, and Future Directions. *Future Gener. Comput. Syst.* **2013**, *29*, 1645–1660. [\[CrossRef\]](#)
- Al-Fuqaha, A.; Guizani, M.; Mohammadi, M.; Aledhari, M.; Ayyash, M. Internet of Things: A Survey on Enabling Technologies, Protocols, and Applications. *IEEE Commun. Surv. Tutor.* **2015**, *17*, 2347–2376. [\[CrossRef\]](#)
- Perera, C.; Zaslavsky, A.; Christen, P.; Georgakopoulos, D. Context Aware Computing for The Internet of Things: A Survey. *IEEE Commun. Surv. Tutor.* **2014**, *16*, 414–454. [\[CrossRef\]](#)
- Borgia, E. The Internet of Things Vision: Key Features, Applications and Open Issues. *Comput. Commun.* **2014**, *54*, 1–31. [\[CrossRef\]](#)
- Miorandi, D.; Sicari, S.; De Pellegrini, F.; Chlamtac, I. Internet of Things: Vision, Applications and Research Challenges. *Ad. Hoc Netw.* **2012**, *10*, 1497–1516. [\[CrossRef\]](#)
- Whitmore, A.; Agarwal, A.; Xu, L.D. The Internet of Things—A Survey of Topics and Trends. *Inf. Syst. Front.* **2015**, *17*, 261–274. [\[CrossRef\]](#)
- Botta, A.; de Donato, W.; Persico, V.; Pescapè, A. Integration of Cloud Computing and Internet of Things: A Survey. *Future Gener. Comput. Syst.* **2016**, *56*, 684–700. [\[CrossRef\]](#)
- Shi, W.; Cao, J.; Zhang, Q.; Li, Y.; Xu, L. Edge Computing: Vision and Challenges. *IEEE Internet Things J.* **2016**, *3*, 637–646. [\[CrossRef\]](#)
- Dastjerdi, A.V.; Buyya, R. Fog Computing: Helping the Internet of Things Realize Its Potential. *Computer* **2016**, *49*, 112–116. [\[CrossRef\]](#)
- Herzig, J.; Nowak, P.K.; Müller, T.; Piccinno, F.; Eisenschlos, J.M. TaPas: Weakly Supervised Table Parsing via Pre-training. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*; Association for Computational Linguistics: Stroudsburg, PA, USA, 2020; pp. 4320–4333. [\[CrossRef\]](#)
- Smock, B.; Pesala, R.; Abraham, R. PubTables-1M: Towards Comprehensive Table Extraction from Unstructured Documents. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*; IEEE: Piscataway, NJ, USA, 2022; pp. 4624–4632. [\[CrossRef\]](#)
- Xu, Y.; Li, M.; Cui, L.; Huang, S.; Wei, F.; Zhou, M. LayoutLM: Pre-training of Text and Layout for Document Image Understanding. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*; ACM: New York, NY, USA, 2020; pp. 1192–1200. [\[CrossRef\]](#)
- Mathew, M.; Karatzas, D.; Jawahar, C.V. DocVQA: A Dataset for VQA on Document Images. In *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision*; IEEE: Piscataway, NJ, USA, 2021; pp. 2199–2208. [\[CrossRef\]](#)

16. Kim, G.; Hong, T.; Yim, M.; Nam, J.; Park, J.; Yim, J.; Hwang, W.; Yun, S.; Han, D.; Park, S. OCR-free Document Understanding Transformer. In *Computer Vision—ECCV 2022*; Springer: Cham, Switzerland, 2022; pp. 498–517. [\[CrossRef\]](#)
17. Lo, K.; Wang, L.L.; Neumann, M.; Kinney, R.; Weld, D.S. S2ORC: The Semantic Scholar Open Research Corpus. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*; ACL: Stroudsburg, PA, USA, 2020; pp. 4969–4983. [\[CrossRef\]](#)
18. Lewis, P.; Perez, E.; Piktus, A.; Petroni, F.; Karpukhin, V.; Goyal, N.; Küttler, H.; Lewis, M.; Yih, W.; Rocktäschel, T.; et al. Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks. In *Advances in Neural Information Processing Systems*; NeurIPS: Sydney, Australia, 2020; Volume 33, pp. 9459–9474.
19. Gao, Y.; Xiong, Y.; Gao, X.; Jia, K.; Pan, J.; Bi, Y.; Dai, Y.; Sun, J.; Wang, M.; Wang, H. Retrieval-Augmented Generation for Large Language Models: A Survey. *arXiv* **2023**, arXiv:2312.10997.
20. Maynez, J.; Narayan, S.; Bohnet, B.; McDonald, R. On Faithfulness and Factuality in Abstractive Summarization. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*; ACL: Stroudsburg, PA, USA, 2020; pp. 1906–1919. [\[CrossRef\]](#)
21. Ji, Z.; Lee, N.; Frieske, R.; Yu, T.; Su, D.; Xu, Y.; Ishii, E.; Bang, Y.J.; Madotto, A.; Fung, P. Survey of Hallucination in Natural Language Generation. *ACM Comput. Surv.* **2023**, *55*, 1–38. [\[CrossRef\]](#)
22. Yao, S.; Zhao, J.; Yu, D.; Du, N.; Shafran, I.; Narasimhan, K.; Cao, Y. ReAct: Synergizing Reasoning and Acting in Language Models. *arXiv* **2023**, arXiv:2210.03629. [\[CrossRef\]](#)
23. Schick, T.; Dwivedi-Yu, J.; Dessì, R.; Raileanu, R.; Lomeli, M.; Zettlemoyer, L.; Cancedda, N.; Scialom, T. Toolformer: Language Models Can Teach Themselves to Use Tools. In *Advances in Neural Information Processing Systems*; NeurIPS: Sydney, Australia, 2023; Volume 36.
24. Asai, A.; Wu, Z.; Wang, Y.; Sil, A.; Hajishirzi, H. Self-RAG: Learning to Retrieve, Generate, and Critique through Self-Reflection. *arXiv* **2024**, arXiv:2310.11511.
25. Guu, K.; Lee, K.; Tung, Z.; Pasupat, P.; Chang, M.W. REALM: Retrieval-Augmented Language Model Pre-Training. In *Proceedings of the 37th International Conference on Machine Learning*; PMLR: London, UK, 2020; Volume 119, pp. 3929–3938.
26. Karpukhin, V.; Oguz, B.; Min, S.; Lewis, P.; Wu, L.; Edunov, S.; Chen, D.; Yih, W. Dense Passage Retrieval for Open-Domain Question Answering. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing*; ACL: Stroudsburg, PA, USA, 2020; pp. 6769–6781. [\[CrossRef\]](#)
27. Izacard, G.; Grave, E. Leveraging Passage Retrieval with Generative Models for Open Domain Question Answering. In *Proceedings of the 16th Conference of the European Chapter of the Association for Computational Linguistics*; ACL: Stroudsburg, PA, USA, 2021; pp. 874–880. [\[CrossRef\]](#)
28. Lyu, Y.; Niu, Z.; Xie, Z.; Zhang, C.; Xu, T.; Wang, Y.; Chen, E. Retrieve-Plan-Generation: An Iterative Planning and Answering Framework for Knowledge-Intensive LLM Generation. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*; Association for Computational Linguistics: Stroudsburg, PA, USA, 2024; pp. 4683–4702. [\[CrossRef\]](#)
29. Baek, I.; Chang, H.; Kim, B.; Lee, J.; Lee, H. Probing-RAG: Self-Probing to Guide Language Models in Selective Document Retrieval. In *Findings of the Association for Computational Linguistics: NAACL 2025*; Association for Computational Linguistics: Stroudsburg, PA, USA, 2025; pp. 3287–3304. [\[CrossRef\]](#)
30. Knollmeyer, S.; Caymazer, O.; Grossmann, D. Document GraphRAG: Knowledge Graph Enhanced Retrieval Augmented Generation for Document Question Answering Within the Manufacturing Domain. *Electronics* **2025**, *14*, 2102. [\[CrossRef\]](#)
31. Devlin, J.; Chang, M.W.; Lee, K.; Toutanova, K. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics*; ACL: Stroudsburg, PA, USA, 2019; pp. 4171–4186. [\[CrossRef\]](#)
32. Beltagy, I.; Lo, K.; Cohan, A. SciBERT: A Pretrained Language Model for Scientific Text. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing*; ACL: Stroudsburg, PA, USA, 2019; pp. 3613–3618. [\[CrossRef\]](#)
33. Huang, Y.; Lv, T.; Cui, L.; Lu, Y.; Wei, F. LayoutLMv3: Pre-training for Document AI with Unified Text and Image Masking. In *Proceedings of the 30th ACM International Conference on Multimedia*; ACM: New York, NY, USA, 2022; pp. 4083–4091. [\[CrossRef\]](#)
34. Han, Z.; Wang, J. Knowledge Enhanced Graph Inference Network Based Entity-Relation Extraction and Knowledge Graph Construction for Industrial Domain. *Front. Eng. Manag.* **2024**, *11*, 143–158. [\[CrossRef\]](#)
35. Karpas, E.; Abend, O.; Belinkov, Y.; Lenz, B.; Lieber, O.; Ratner, N.; Shoham, Y.; Bata, H.; Levine, Y.; Leyton-Brown, K.; et al. MRKL Systems: A Modular, Neuro-Symbolic Architecture That Combines Large Language Models, External Knowledge Sources and Discrete Reasoning. *arXiv* **2022**, arXiv:2205.00445.
36. Gao, L.; Madaan, A.; Zhou, S.; Alon, U.; Liu, P.; Yang, Y.; Callan, J.; Neubig, G. PAL: Program-Aided Language Models. In *Proceedings of the 40th International Conference on Machine Learning*; PMLR: London, UK, 2023; Volume 202, pp. 10764–10799.
37. Huang, S.; Zhang, M.; Hu, B.; Zhang, M. ToolOmni: Enabling Open-World Tool Use via Agentic Learning with Proactive Retrieval and Grounded Execution. In *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics*; Association for Computational Linguistics: San Diego, CA, USA, 2026; pp. 37421–37439. [\[CrossRef\]](#)

38. Thorne, J.; Vlachos, A.; Christodoulopoulos, C.; Mittal, A. FEVER: A Large-scale Dataset for Fact Extraction and VERification. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics*; ACL: Stroudsburg, PA, USA, 2018; pp. 809–819. [\[CrossRef\]](#)
39. Wadden, D.; Lin, S.; Lo, K.; Wang, L.L.; van Zuylen, M.; Cohan, A.; Hajishirzi, H. Fact or Fiction: Verifying Scientific Claims. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing*; ACL: Stroudsburg, PA, USA, 2020; pp. 7534–7550. [\[CrossRef\]](#)
40. Zanella, A.; Bui, N.; Castellani, A.; Vangelista, L.; Zorzi, M. Internet of Things for Smart Cities. *IEEE Internet Things J.* **2014**, *1*, 22–32. [\[CrossRef\]](#)
41. Lin, J.; Yu, W.; Zhang, N.; Yang, X.; Zhang, H.; Zhao, W. A Survey on Internet of Things: Architecture, Enabling Technologies, Security and Privacy, and Applications. *IEEE Internet Things J.* **2017**, *4*, 1125–1142. [\[CrossRef\]](#)
42. *IEEE Std 2413-2019*; IEEE Standard for an Architectural Framework for the Internet of Things. IEEE: Piscataway, NJ, USA, 2020. [\[CrossRef\]](#)
43. Khattab, O.; Zaharia, M. ColBERT: Efficient and Effective Passage Search via Contextualized Late Interaction over BERT. In *Proceedings of the 43rd International ACM SIGIR Conference on Research and Development in Information Retrieval*; ACM: New York, NY, USA, 2020; pp. 39–48. [\[CrossRef\]](#)
44. Mallen, A.; Asai, A.; Zhong, V.; Das, R.; Khashabi, D.; Hajishirzi, H. When Not to Trust Language Models: Investigating Effectiveness of Parametric and Non-Parametric Memories. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics*; Association for Computational Linguistics: Toronto, ON, Canada, 2023; pp. 9802–9822. [\[CrossRef\]](#)
45. Liu, N.F.; Lin, K.; Hewitt, J.; Paranjape, A.; Bevilacqua, M.; Petroni, F.; Liang, P. Lost in the Middle: How Language Models Use Long Contexts. *Trans. Assoc. Comput. Linguist.* **2024**, *12*, 157–173. [\[CrossRef\]](#)
46. Es, S.; James, J.; Espinosa Anke, L.; Schockaert, S. RAGAs: Automated Evaluation of Retrieval Augmented Generation. In *Proceedings of the 18th Conference of the European Chapter of the Association for Computational Linguistics: System Demonstrations*; Association for Computational Linguistics: St. Julian's, Malta, 2024; pp. 150–158. [\[CrossRef\]](#)

Disclaimer/Publisher's Note: The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.