technologies

Article

Intelligent Construction of LVC Resource Interface Protocol Templates Using Large Language Models

Dongfang Wang [1,†], Yusheng Zhang [1,†], Guobao Dong [2,†], Yonghui Xu [1,†], Yu Huang [1,†], Baodi Xie [2,†] and Changan Wei [1,*]

[1] School of Electronics and Information Engineering, Harbin Institute of Technology, Harbin 150001, China; 25s105177@stu.hit.edu.cn (D.W.); 15762425608@163.com (Y.Z.); xyh@hit.edu.cn (Y.X.); 23s105136@stu.hit.edu.cn (Y.H.)
[2] Modeling and Simulation of Complex Systems, National Key Laboratory, Beijing Simulation Center, Beijing 100854, China; dongguobao464@163.com (G.D.); xiebaodi@163.com (B.X.)
* Correspondence: weichangan@hit.edu.cn
† These authors contributed equally to this work.

Abstract

The construction of resource interface protocol templates is a key prerequisite for the unified integration of live, virtual, and constructive (LVC) resources in complex simulation and test environments. However, real-world protocol documents are usually heterogeneous in format, inconsistent in description, and rich in nested structures and implicit semantics, which makes manual analysis inefficient and error-prone. To address this issue, this paper proposes an intelligent construction method for LVC resource interface protocol templates based on large language models. First, raw protocol documents are converted into a unified Markdown representation, and a semantic understanding module is used for main-table identification, minimum-unit splitting, and auxiliary-table association. Then, a protocol item type identification expert module is designed to recognize complex structures such as frame headers, ordinary items, dynamic items, struct items, branch items, sub-protocol items, and checksum items. Finally, the extracted information is integrated into structured intermediate results for automatic XML template generation. Experiments on a representative test set composed of 20 protocol tables from real-world LVC resource interface documents show that the proposed method achieves a main-table extraction accuracy of 0.9761, a type recognition F1-score of 0.9769, an XML generation success rate of 1.0, and a node consistency of 0.9478. These results demonstrate that the proposed method can effectively improve the automation and engineering applicability of protocol template construction.

Keywords: large language model; resource interface protocol; protocol template construction; semantic understanding; XML generation; LVC

Academic Editors: Liviu Marian Ungureanu, Iulian Sorin Munteanu, Péter Galambos and Sotirios K. Goudos

Received: 11 April 2026
Revised: 14 May 2026
Accepted: 21 May 2026
Published: 28 May 2026

1. Introduction

With the rapid development of simulation and testing technologies, live, virtual, and constructive (LVC) resources have been widely used in complex battlefield environment construction, system-of-systems confrontation simulation, and system-level functional verification. To achieve collaboration and interoperability among different resources, various sensors, platforms, command systems, and simulation nodes need to be connected to a unified test system architecture [1]. However, external resources usually adopt proprietary communication protocols predefined by different vendors, whereas data exchange within

the test system relies on a common object model and a publish/subscribe mechanism. The differences between the two sides in terms of data format, structural organization, and semantic representation make direct interconnection difficult. Therefore, an efficient, accurate, and scalable protocol adaptation method is urgently needed to support the rapid integration of heterogeneous resources.

At present, the construction of resource interface protocol templates still mainly relies on manual work. Specifically, operators need to read protocol specification documents in formats such as .doc, .docx, or .pdf, extract protocol field names, byte lengths, data types, units, remarks, and auxiliary-table references item by item, and then organize them into standardized protocol files according to predefined template specifications. This process usually involves multiple steps, including main-table identification, auxiliary-table association, field splitting, type determination, and XML node mapping. It is not only time-consuming and labor-intensive, but also prone to omissions, ambiguities, and structural errors in complex protocol scenarios [2]. As protocol scales continue to increase, nested structures become more complex, and protocol descriptions become more diverse, traditional manual approaches can no longer satisfy the engineering requirements for high efficiency, high consistency, and large-scale processing [3].

To address these issues, previous studies have attempted to use rule matching, regular expressions, and traditional natural language processing techniques to automatically process protocol texts [4]. These methods can achieve acceptable performance in scenarios with relatively standardized formats and fixed descriptions. However, when protocol documents contain complex situations such as cross-table references, semantic supplements in remarks, bit-definition structures, dynamically repeated fields, and nested sub-protocols, they often fail to work reliably [5]. In recent years, large language models have shown strong capabilities in table understanding, technical document analysis, and complex reasoning tasks, thereby providing a new technical path for the automatic understanding of protocol documents [6–9]. Nevertheless, resource interface protocols are not ordinary natural language texts. Their processing requires not only semantic understanding, but also the generation of intermediate representations that satisfy the input requirements of downstream XML template construction programs under engineering constraints [10,11]. Therefore, how to effectively combine the semantic capability of large language models with protocol-domain knowledge, structured constraints, and downstream template generation procedures is the key issue for the intelligent construction of resource interface protocol templates [12].

To solve the above challenges, this paper proposes an intelligent construction method for resource interface protocol templates based on large language models, enabling the automatic conversion from raw protocol documents to standardized XML digital mapping model files. Based on protocol preprocessing and semantic understanding, the proposed method constructs a staged processing pipeline for engineering scenarios by introducing protocol item type identification experts, structured information integration, and XML template generation modules. Different from methods that directly rely on one-shot generation by a large language model, this work emphasizes the explicit linkage among protocol understanding, type determination, and template generation, thereby improving the stability and deployability of the system in complex protocol scenarios.

Compared with general LLM-based document understanding and structured information extraction frameworks, the proposed method is not designed as a generic extraction pipeline. Instead, it is oriented toward executable protocol-template construction under engineering constraints. The key difference is that the proposed approach introduces a protocol-item taxonomy, expert-level task decomposition, constrained intermediate representations, and deterministic XML generation. In this way, the LLM is not used as an

unconstrained end-to-end generator, but as a semantic reasoning component embedded in a structured protocol-construction workflow.

The main contributions of this paper are summarized below.

First, from the perspective of methodological modeling, a protocol item taxonomy for LVC resource interface protocols is established. Based on more than 30 protocol specification documents and over 5000 protocol items, the taxonomy covers frame headers, ordinary items, dynamic items, struct items, branch items, sub-protocol items, checksum items, and frame tails, and further refines ordinary items into raw fields, physical values, BCD codes, ASCII codes, bit-field parsing items, and enumeration items.

Second, from the perspective of LLM-based semantic reasoning, an expert prompt decomposition strategy is designed for protocol semantic understanding and protocol item type identification. Instead of relying on one-shot generation, the proposed method decomposes the task into main-table identification, minimum-unit splitting, auxiliary-table association, and type-specific expert recognition, thereby improving the controllability of LLM outputs in complex protocol scenarios.

Third, from the perspective of structured representation, a constrained intermediate representation mechanism is proposed. The nine-column semantic table provides normalized field-level attributes, while the protocol item type and structural description result provides higher-level organization information such as dynamic grouping, branch conditions, nested sub-protocols, and checksum constraints. These two intermediate results jointly bridge LLM-based semantic understanding and downstream XML template generation.

Fourth, from the perspective of engineering integration, the proposed pipeline connects the above methodological components with a deterministic XML template generation program. Experiments on a test set composed of 20 protocol-table samples show that the proposed method outperforms rule-based and single-stage large language model baselines in terms of main-table extraction accuracy, type-recognition F1-score, XML generation success rate, and node consistency.

The remainder of this paper is organized as follows. Section 2 introduces the protocol dataset and the protocol item taxonomy. Section 3 presents the overall methodological framework and key modules. Section 4 validates the effectiveness of the proposed method through multiple experiments and provides ablation analysis, prompt sensitivity and cost analysis, and an illustrative case study. Section 5 discusses the experimental findings and the limitations of the method. Section 6 concludes the paper and outlines future work.

2. Protocol Dataset and Protocol Item Taxonomy

2.1. Protocol Dataset

To systematically analyze the compositional characteristics and structural patterns of resource interface protocols, this paper collects real protocol specification documents from multiple research institutions and industrial collaborators, covering a variety of LVC resources, such as sensors, platforms, control devices, simulation nodes, and their supporting communication interfaces. In total, more than 30 protocol specification documents are collected. These documents involve different vendors, interface standards, and engineering scenarios and therefore exhibit strong heterogeneity and representativeness.

The data providers were selected according to three criteria. First, they should have practical experience in LVC resource integration, simulation-test system construction, or equipment interface development. Second, the provided protocol documents should contain complete field-level descriptions, including field names, byte lengths, data types, remarks, and, where applicable, auxiliary tables or bit-definition tables. Third, the collected documents should cover different resource types and communication-interface styles so that the dataset could support the analysis of heterogeneous protocol structures.

In this paper, a "protocol document" refers to a complete protocol specification file, a "protocol table" refers to a field-definition table within a protocol document, and a "protocol item" refers to a field-level or structure-level unit extracted from a protocol table.

Structurally, the collected protocol documents usually contain one or more main protocol tables, together with supplementary materials such as remarks, value-description tables, bit-definition tables, code tables, and nested sub-protocol descriptions. Based on these protocol specification documents, more than 5000 protocol items are extracted and analyzed. Overall, these protocols show significant differences in field definition methods, structural organization, and semantic expression styles. On the one hand, they contain fixed structures with clear rules, such as frame headers, checksum fields, and frame tails, as well as conventional field types, such as physical values, BCD encoding, bit-field parsing, and strings. On the other hand, they also include more complex organizational forms, such as dynamically repeated fields, conditional branches, struct arrays, nested sub-protocols, and supplementary explanations provided in external tables. These characteristics indicate that resource interface protocol processing is not merely a field extraction task, but rather a comprehensive modeling problem that needs to consider structure, semantics, and engineering constraints simultaneously.

To support subsequent method design and experimental validation, a protocol benchmark test set was further constructed, and samples are organized and partitioned at the protocol-table level. The reason for adopting table-level partitioning instead of protocol-item-level partitioning is that documents usually exhibit strong structural consistency and expression-style correlation internally. If random splitting were performed at the field level, information leakage could easily occur, leading to overestimated model performance. Therefore, all subsequent experiments in this paper use protocol tables as the basic evaluation unit, and protocol understanding, structured extraction, and XML template generation performance are all measured at the table level.

Due to confidentiality requirements in real-world engineering projects, project names, device names, vendor-specific identifiers, and sensitive field contents were anonymized or replaced where necessary. However, the structural information required for this study, including field organization, table layout, data types, byte lengths, references to auxiliary tables, parsing rules, and XML mapping requirements, was preserved. This processing does not affect the analysis of protocol structures or the evaluation of the proposed method.

For experimental evaluation, 20 protocol-table samples were selected from the collected protocol tables to construct the final benchmark test set. The samples were selected according to four criteria: protocol-item type coverage, header and layout heterogeneity, structural complexity, and representative presentation value. Specifically, the selected samples were required to cover both common protocol fields and complex protocol structures, including frame headers, raw fields, physical-value fields, BCD-coded fields, ASCII-coded fields, bit-field parsing fields, enumeration fields, string fields, dynamic items, struct items, branch items, sub-protocol items, checksum fields, and frame tails. The samples include protocol tables originating from both different protocol documents and different tables within the same document. This design was adopted because some real protocol documents contain multiple heterogeneous tables with different structures. Therefore, the benchmark was organized at the protocol-table level rather than at the protocol-document level.

To provide a clearer overview of the collected protocol data, Table 1 summarizes the contextual characteristics of the protocol dataset. The statistics focus on structural and semantic characteristics relevant to protocol understanding and XML template generation.

Table 1. Contextual characteristics of the collected protocol dataset.

Category	Item	Description
Source	Data provider	Real-world LVC resource interface protocol documents provided by multiple research institutions and industrial collaborators.
Source	Application context	LVC resource integration, simulation-test system construction, and equipment interface development.
Scale	Protocol documents	More than 30 protocol specification documents.
Scale	Analyzed protocol items	More than 5000 field-level or structure-level protocol items.
Resource type	Covered resources	Sensors, platforms, control devices, simulation nodes, and supporting communication interfaces.
Document structure	Typical table types	Main protocol tables, supplementary value-description tables, bit-definition tables, code tables, branch tables, and nested sub-protocol tables.
Field semantics	Typical field attributes	Field name, byte length, data type, signedness, unit, scale factor, value range, default value, remarks, and referenced table.
Data representation	Typical data forms	Raw numerical fields, physical-value fields, BCD-coded fields, ASCII-coded fields, enumeration fields, string fields, and bit-field parsing fields.
Structural complexity	Typical complex structures	Frame headers, frame tails, checksum fields, dynamic repeated fields, struct arrays, conditional branches, and nested sub-protocols.
Evaluation design	Evaluation unit	Protocol table level rather than field level, to avoid information leakage caused by field-level random splitting.

As shown in Table 1, the collected dataset contains not only conventional scalar fields but also multiple types of complex protocol structures. Therefore, the dataset supports the analysis of both field-level semantic normalization and structure-level protocol-template generation. The benchmark construction in this study focuses on preserving these structural characteristics, because they directly determine the difficulty of main-table identification, auxiliary-table association, protocol item type recognition, and XML template generation.

2.2. Design of the Protocol Item Taxonomy

Based on the collected protocol specification documents and the systematic analysis of more than 5000 protocol items, this paper constructs a hierarchical protocol item taxonomy for resource interface protocols, as shown in Figure 1. The protocol item taxonomy was constructed through an iterative analysis and expert-validation process. First, more than 5000 protocol items extracted from the protocol tables were manually inspected to identify recurring field roles, parsing behaviors, and structural organization patterns. Second, protocol items with similar processing logic were grouped into preliminary categories. For example, fields with units and scale factors were grouped as physical value items, fields described by bit ranges or status flags were grouped as bit-field parsing items, and fields whose occurrence counts depended on preceding length or count fields were grouped as dynamic items. Third, these preliminary categories were further refined according to the requirements of downstream XML template generation, because the taxonomy in this study is intended not only for semantic description but also for executable protocol template construction. Finally, the category definitions were reviewed by domain experts involved in protocol analysis and template construction, and the coverage of the taxonomy was checked against the collected protocol tables.

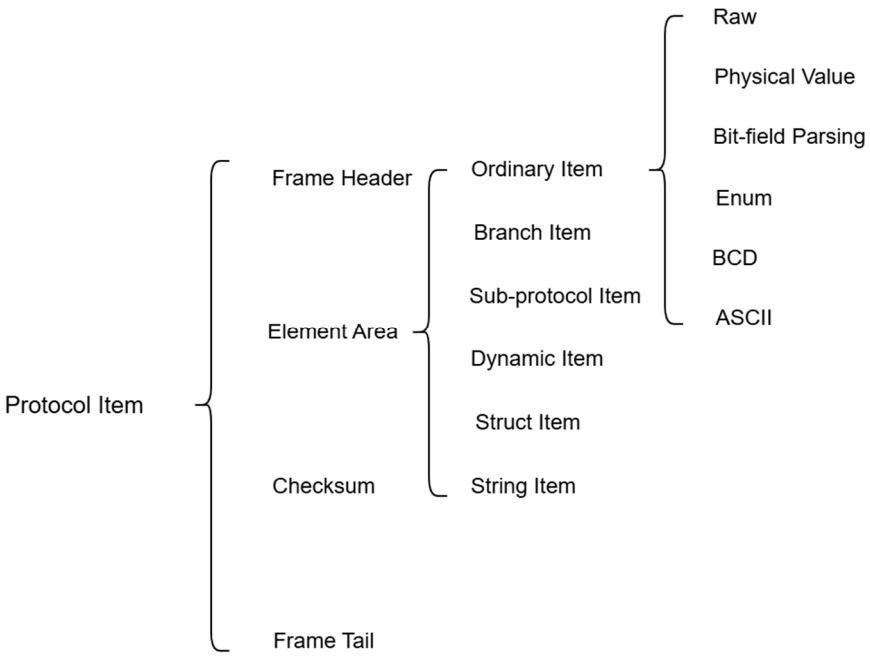


Figure 1. Protocol item taxonomy.

It should be noted that the proposed taxonomy is not intended to cover all possible protocol structures across industrial domains. Rather, it is designed to characterize the major protocol item types observed in the collected LVC resource interface protocol tables and to support the XML template generation task investigated in this study. The taxonomy provides a unified representation framework for the structured description of heterogeneous protocol documents and supports subsequent tasks such as protocol semantic understanding, type identification, and XML template generation. For an actual protocol table, a protocol frame usually consists of four parts, namely, a frame header, an element area, a checksum item, and a frame tail. Among them, the frame header and frame tail are optional structural components, the checksum item is used to ensure transmission integrity, and the element area carries the core semantic content of the protocol and is therefore the main focus of the taxonomy constructed in this work.

As shown in Figure 1, the element area is further divided into six major protocol item categories, including ordinary items, branch items, sub-protocol items, dynamic items, struct items, and string items. This classification not only covers common scalar fields, but also describes condition branches, nested protocol blocks, repeated structures, and textual fields that frequently occur in engineering documents. Compared with a simple flat label system, this hierarchical organization is better suited to expressing the complexity of resource interface protocols in terms of structural organization and parsing behavior. From a more general methodological perspective, the construction of such a protocol item taxonomy can also be viewed as a taxonomy inference problem for table semantics. Recent studies have begun to explore the use of large language models to infer entity types and hierarchical relations from tabular data [13].

Among these categories, ordinary items correspond to the most common type of fields in protocols and usually represent atomic data units that can be directly mapped to individual XML elements. According to differences in parsing and processing behavior, ordinary items can be further divided into several processing-oriented subtypes, including raw, physical value, bit-field parsing, enumeration, BCD, and ASCII. Specifically, the raw type indicates that the field content can be used directly without additional transformation. The physical value type needs to be converted into engineering-meaningful numerical values

according to rules such as scale factors and offsets. The bit-field parsing type is suitable for cases in which multiple semantic states are compressed into the same binary field. The enumeration type is used to represent discrete states or symbolic meanings. BCD and ASCII correspond to decimal-coded content and character-coded content, respectively. Through this fine-grained categorization, ordinary items can describe not only data representation forms but also their corresponding parsing logic in a unified manner.

Branch items are used to describe scenarios in which the subsequent protocol structure depends on the value of a preceding control field. In such cases, different values usually correspond to different field combinations, message-body structures, or parsing rules. Sub-protocol items are used to represent nested protocol blocks with relatively independent internal organization and are suitable for describing situations in which a protocol item itself contains another protocol structure. These two types are particularly common in practical engineering applications, such as mode-related payloads, command-specific message bodies, and communication structures with recursive nesting characteristics.

Dynamic items are used to describe repeated fields whose occurrence count or occupied length can only be determined at runtime. Such protocol items are usually composed of one or more groups of dynamic elements, and their repetition counts are often constrained by contextual fields such as length fields or count fields. Struct items are similar to dynamic items in that they also organize multiple fields into an integrated whole. However, their focus is on expressing repeated structural records with relatively fixed internal layouts, such as measurement block arrays and parameter tuple sequences. Compared with ordinary items, both dynamic items and struct items belong to higher-level structural units and are important components for representing repeated patterns in protocol templates.

String items are used to carry textual or identifier-like information, such as device names, command strings, diagnostic messages, and version descriptions. In actual protocol specifications, such fields are typically encoded as multi-byte character sequences and may also appear in variable-length form. Since their semantic role differs significantly from that of numerical fields, they are modeled as an independent protocol item type in this work.

Overall, the protocol item taxonomy proposed in this paper not only covers common basic protocol fields, but also represents complex engineering structures such as dynamic repetition, conditional branching, structural nesting, and sub-protocol reuse. Therefore, it provides a unified representation basis for subsequent large language model-based protocol semantic understanding, protocol item type identification, and structured XML generation.

To make the taxonomy construction basis more explicit, Table 2 summarizes the main identification evidence and XML mapping role of each protocol item type. The table shows that the taxonomy is organized according to both semantic evidence in protocol tables and the structural requirements of downstream XML template generation. For readability and generality, the XML mapping roles are described using generalized English terms rather than project-specific implementation tags.

For example, a field described with a unit, scale factor, or value range, such as an angle field with a resolution of 0.01°, is categorized as a physical value item because it requires numerical conversion during template generation. A field described by bit positions or bit ranges, such as "Bit0: start flag; Bit1: fault flag", is categorized as a bit-field parsing item because its semantic content must be expanded into bit-level definitions. A field whose value controls the number of following repeated records, such as a target count field, is categorized as a dynamic item. A field whose value determines different subsequent parsing paths is categorized as a branch item, while a field that refers to another relatively independent protocol table is categorized as a sub-protocol item.

Table 2. Protocol item taxonomy, identification evidence, and XML mapping role.

Protocol Item Type	Main Identification Evidence	Typical Role in Template Generation
Frame header	Located at the beginning of a frame; fixed value; keywords such as frame header, synchronization word, or frame ID	Mapped to frame-header nodes
Raw item	Field has byte length and data type but no special decoding rule	Mapped to a basic field node without additional processing
Physical value	Unit, scale factor, offset, signedness, physical meaning, or value range	Mapped to a field node with conversion attributes
BCD item	BCD keyword or decimal-coded time/value description	Mapped to a BCD-decoding field node
ASCII item	ASCII keyword or character-coded identifier	Mapped to an ASCII-decoding field node
Bit-field parsing item	Bit positions, bit ranges, status flags, or bit-definition descriptions	Mapped to bit-group and bit-value definition nodes
Enumeration item	Discrete value-description mappings	Mapped to enumeration-definition and enumeration-value nodes
String item	Device name, version text, command string, or character sequence	Mapped to a string field node
Dynamic item	Count-controlled or variable repeated elements	Mapped to a repeated field group controlled by a count or length field
Struct item	A group of fields forming repeated records with a fixed internal layout	Mapped to a structured array node with internal child fields
Branch item	A control field determines subsequent fields or parsing paths	Mapped to conditional branch nodes with different jump values
Sub-protocol item	Reference to another independent protocol table or nested protocol body	Mapped to nested protocol nodes
Checksum item	Checksum, CRC, parity, or sum-check keyword and checksum range	Mapped to checksum attributes or checksum field nodes
Frame tail	Located at the end of a frame; fixed end marker	Mapped to frame-tail nodes

3. Materials and Methods

3.1. System Architecture

This paper constructs an automated intelligent system for resource interface protocols to realize the automatic conversion from raw protocol text to digital mapping model files in XML format. The overall system architecture is shown in Figure 2 and mainly consists of five stages: protocol preprocessing, large language model-based semantic understanding, protocol item type identification, information extraction and integration, and XML template generation. First, the preprocessing module converts raw protocol documents into a unified Markdown format to reduce the influence of differences in document format, page layout, and typesetting style on subsequent processing. The preprocessed protocol text is then fed into the semantic understanding module, where the main tables, auxiliary tables, remarks, and cross-table associations in the protocol document are jointly analyzed to generate standardized semantic table results. Similar findings have also been reported in layout-aware generative document understanding studies, showing that explicitly incorporating spatial layout information into language model reasoning can effectively improve information organization and generation quality in complex document scenarios [14,15]. On this basis, the protocol item type identification expert module further determines the

protocol item type of each field and its structural relationships. Subsequently, the system integrates semantic understanding results with various type recognition results to produce structured descriptions for XML generation. Finally, an XML protocol conversion program automatically generates XML files conforming to the digital mapping model specification based on the above results, thereby completing the intelligent construction of resource interface protocol templates.

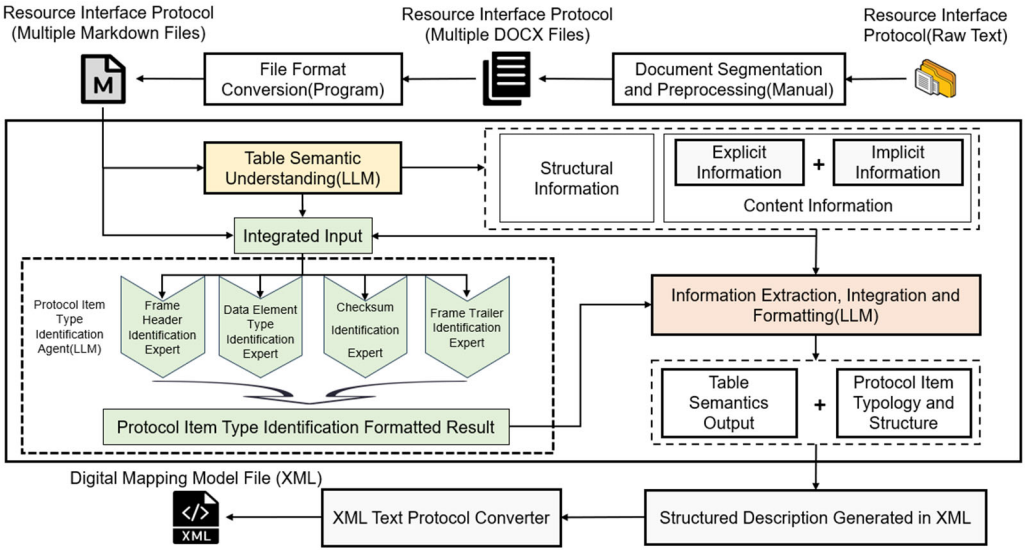


Figure 2. Architecture of the automated intelligent construction system for resource interface protocols.

3.2. Intelligent Protocol Processing Based on Large Language Models

3.2.1. Semantic Understanding Based on Large Language Models

Resource interface protocols are usually organized in tabular form. However, actual documents often contain heterogeneous information simultaneously, including table headers, remarks, auxiliary tables, code tables, and bit-definition tables. Therefore, protocol content is not only highly structured but also contains a large amount of implicit semantics that can only be determined in context [16]. To address this issue, this paper introduces a semantic understanding module based on large language models. Using Markdown-formatted protocol text as a unified input, the large language model reads, reasons over, and structurally organizes protocol tables, thus converting raw heterogeneous tables into standardized semantic representations. This process is not a simple field extraction procedure; instead, it identifies the actual meaning of fields, their hierarchical relationships, and the evidence links between remarks and auxiliary tables through semantic analysis, thereby elevating protocol processing from traditional syntax-level matching to semantic-level understanding [17].

As shown in Figure 3, the semantic understanding process mainly includes three steps. First, the model identifies the main table in the full text, that is, the table that provides item-by-item field descriptions, and merges the remaining explanatory tables or text blocks into auxiliary tables or supplementary blocks to ensure the uniqueness of the subsequent processing target. Second, the main table is split at the minimum-unit level and uniformly renumbered. Here, the minimum unit refers to the leaf-level field that should appear as an independent node in the subsequent template generation stage. For multiple fields appearing in parallel within the same cell, the model splits them according to predefined rules and establishes a continuous and consistent row numbering system to eliminate the interference caused by cross-page layout, repeated numbering, and hierarchical numbering. Third, the model establishes bidirectional association relationships among the main table, auxiliary tables, and remarks. On the one hand, it identifies explicit references in the main

table. On the other hand, it reversely locates the corresponding fields based on auxiliary-table titles, introductory statements, and referential expressions in remarks, thus forming explicit mappings between main-table fields and explanatory evidence. This strategy of first locating the main structure, then splitting minimum units, and finally establishing cross-block associations is consistent with the idea of first performing block-level organization and then carrying out local reasoning in complex visual documents, and it helps reduce semantic drift during cross-region information fusion [18,19].

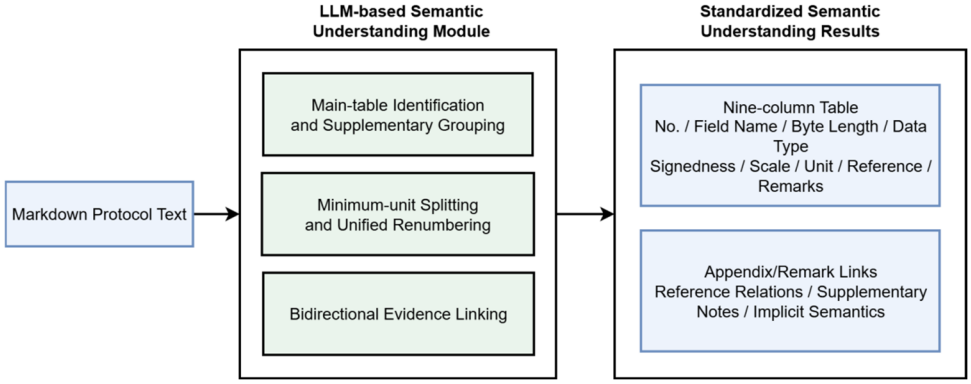


Figure 3. Workflow of the LLM-based semantic understanding module.

To ensure stable invocation by downstream modules, the output format of the semantic understanding stage is uniformly constrained in this work. The result is organized into a standardized table with a fixed nine-column header, namely, "No.", "Information Name", "Byte Length", "Data Type", "Signed/Unsigned", "Scale Factor", "Unit", "Referenced Table", and "Remarks". The nine-column schema was designed according to two principles. First, it preserves the field-level information that repeatedly appears in heterogeneous protocol tables. For example, field order, field name, field length, data type, signedness, scale factor, unit, referenced table, and remarks are common types of information in the collected protocol documents, although they may be expressed using different headers or layouts.

Second, the schema provides the minimum information required by the subsequent protocol item identification and XML template generation modules. The columns "No.", "Information Name", "Byte Length", "Data Type", and "Signed/Unsigned" support basic XML element construction. The columns "Scale Factor" and "Unit" support the recognition of physical-value fields. The column "Referenced Table" preserves cross-table links, and the column "Remarks" retains additional evidence such as fixed values, enumeration meanings, bit definitions, dynamic repetition rules, and checksum rules. Therefore, these heterogeneous descriptions are normalized into a unified nine-column intermediate table.

Specifically, "No." records the normalized order of each minimum field unit in the protocol frame; "Information Name" stores the semantic name of the field; "Byte Length", "Data Type", and "Signed/Unsigned" describe the basic data attributes required for XML element construction; "Scale Factor" and "Unit" store quantization, scaling, and physical-unit information for identifying physical-value fields; "Referenced Table" preserves explicit or implicit links between the main table and auxiliary tables, such as bit-definition tables, branch tables, or sub-protocol tables; and "Remarks" stores value ranges, fixed values, enumeration meanings, bit-level definitions, dynamic repetition rules, checksum rules, and other supplementary descriptions that cannot be placed in the previous structured columns. This result retains not only the explicit field attributes in the main table, but also implicit semantic information derived from auxiliary tables and remarks. In engineering implementation, the semantic understanding stage first generates an intermediate representation containing the association clues between the main table and auxiliary tables,

and a dedicated extraction stage then outputs the nine-column main-table result, thereby providing a unified and reliable input basis for subsequent protocol item type identification and structural integration.

3.2.2. Construction of Protocol Item Type Identification Experts

After semantic understanding, the system obtains a unified intermediate representation. This representation contains three types of information: the row numbers of minimum units in the main table, explicit field attributes, and supplementary semantics extracted from remarks and referenced tables. The next task is to infer the parsing role of each unit for template generation. In this study, this parsing role corresponds to the protocol item type.

Unlike byte length or data type, the protocol item type is rarely stated directly in the original document. It must be inferred from field names, descriptions, reference relationships, surrounding context, and engineering knowledge. To solve this problem, this paper constructs a type-identification expert module. The module organizes domain discrimination rules, structural constraints, and representative examples into reusable expert prompts. Guided by these prompts, the LLM reasons over the current protocol table, identifies field roles and structural relationships, and outputs annotations that can be parsed by downstream modules.

As shown in Figure 4, the expert module uses two inputs: the intermediate representation from the semantic understanding stage and the original Markdown protocol text. It classifies protocol items through an "expert prompt constraints + LLM judgment" mechanism. Each prompt specifies identification criteria, trigger features, output-format constraints, and representative examples. The LLM then infers field roles and nesting relationships and produces structured outputs for downstream parsing. Finally, the outputs from different experts are aggregated into a structural description for XML generation.

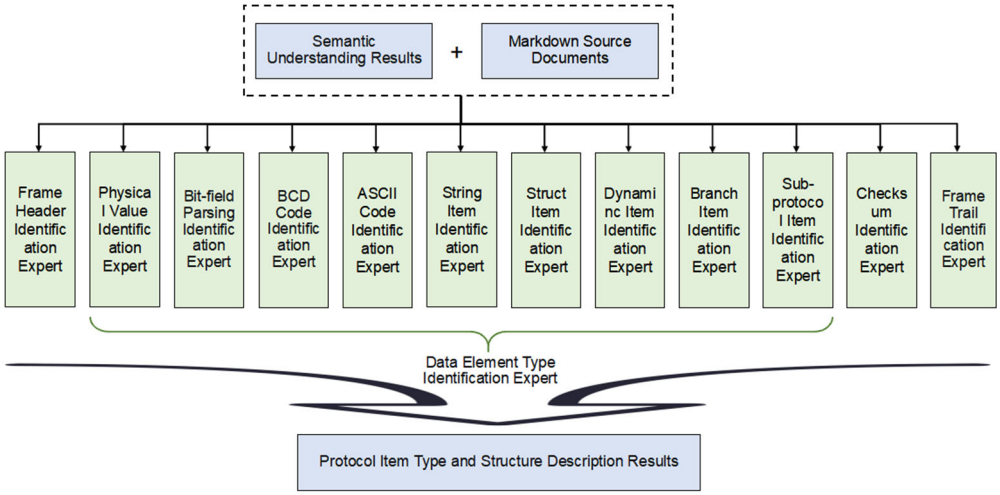


Figure 4. Workflow of the protocol item type identification expert.

Based on the previous analysis of more than 30 protocol specification documents and over 5000 protocol items, the protocol item labels used in engineering implementation include frame header, raw, physical value, bit-field parsing, BCD encoding, ASCII encoding, enumeration, string, dynamic item, dynamic element, struct item, struct element, branch item, branch element, sub-protocol item, sub-protocol element, checksum item, and frame tail. Among them, frame headers, frame tails, and checksum items are mainly recognized according to field positions and explicit semantic trigger words. Ordinary field processing forms such as physical value, BCD encoding, ASCII encoding, bit-field parsing, and enumeration rely more on local evidence such as units, remarks, value descriptions,

and bit-range definitions. Complex structural types such as dynamic items, struct items, branch items, and sub-protocol items require comprehensive reasoning based on cross-row relations, auxiliary-table references, and contextual organization. This label system not only covers common protocol elements, but also effectively describes complex engineering structures such as dynamic repetition, conditional branching, and nested protocols.

To improve the stability and interpretability of the identification process, protocol item type identification is further decomposed into several collaborative expert subtasks, such as frame header recognition, ordinary field processing-form determination, bit-field parsing and enumeration recognition, dynamic-item and struct-item boundary identification, branch-item and branch-path identification, sub-protocol recognition, checksum identification, and frame-tail recognition. Each subtask adopts the output form of "type identification + key information summarization", which not only reports the set of main-table row numbers covered by the current type, but also outputs structural information closely related to that type to support subsequent information extraction and XML generation. For example, branch-item and sub-protocol recognition tasks further output the corresponding groups, jump values, or sub-protocol table information. Struct recognition tasks provide struct-element grouping and repetition counts. Bit-field parsing and enumeration recognition tasks supplement bit-range definitions or the meanings of enumerated values. Through this subtask decomposition and structured aggregation mechanism, the type recognition results become not just simple labels, but structured judgments that can be directly invoked by downstream modules.

In addition, considering that protocol documents often contain multi-level nesting, cross-table references, and coexistence of multiple labels, this work allows for the same field to play composite semantic roles at different levels. For example, a field may belong to a bit-field parsing item among ordinary fields while also serving as the basis for branch determination in branch structure organization. Some fields may further expand into sub-protocol or bit-definition structures through referenced tables. To handle such complex scenarios, the protocol item type identification experts inherit the main-table–auxiliary-table association clues established in the semantic understanding stage and perform unified aggregation and consistency refinement based on multiple expert outputs, so that type determination is constrained not only by local textual evidence but also by the overall structural logic of the protocol. Ultimately, this module outputs the row-number sets corresponding to different protocol item types together with the necessary structural description information, thus providing direct support for the next-stage information extraction, integration, and XML template generation.

3.2.3. Information Extraction, Integration, and Formatted Output

After table semantic understanding and protocol item type identification are completed, the system has obtained two categories of key results. One is the nine-column main-table result, which provides field-level explicit attributes. The other is the type recognition result produced by each protocol item identification expert, which specifies the semantic category and organizational relationship of different fields in the protocol structure. The goal of the third stage is to integrate these two types of information within a unified context and transform the scattered, heterogeneous, and hierarchically uneven content in raw protocol tables into normalized descriptions for XML template generation. This stage is not a simple concatenation of existing results. Instead, it performs unified extraction, summarization, and organization of field attributes, structural relationships, and additional semantics according to the needs of subsequent template filling, thereby producing structured outputs that can be directly parsed and invoked by downstream programs.

As shown in Figure 5, the information extraction, integration, and formatted output module takes as its core inputs the large language model results from the previous two stages: the nine-column main-table information produced in the semantic understanding stage and the type identification and structural description information generated by the protocol recognition experts. Under the joint effect of large language model reasoning and format constraints, the system integrates these inputs in a unified manner, thereby avoiding inconsistencies caused by repeated inference on the same field in subsequent processing. Different integration strategies are adopted for different protocol item types. For ordinary items, attributes such as data type, scale factor, and processing function are retained. For physical value items, additional information such as multiplier, offset, and target unit is supplemented. For bit-field parsing items, bit ranges, enumerated meanings, and auxiliary-table reference relationships are added. For dynamic items, branch items, and sub-protocol items, internal element boundaries, grouping relations, and nested structural information are also preserved.

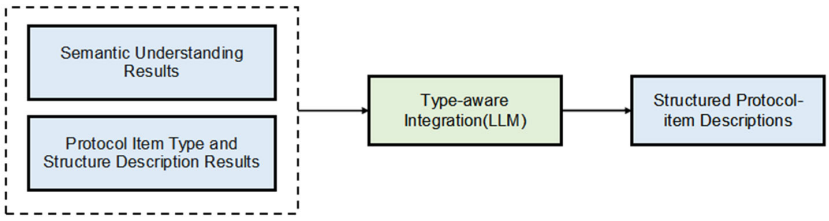


Figure 5. Workflow of the information integration and formatted output module.

In engineering implementation, different expert modules in the previous stage separately output the recognition results of various protocol item types. In the current stage, a unified integration prompt aggregates these results together with the nine-column main table to form a normalized textual description of protocol item types and structures. This result preserves the order, labels, and hierarchical relationships required by program parsing and serves, together with the nine-column main table, as the direct input to the subsequent XML generation module. In this way, the outputs of large language models are transformed from raw natural-language understanding results into standardized intermediate results that can be directly consumed by the program side, enabling stable linkage from semantic understanding and type identification to template generation and providing a consistent, controlled, and reproducible data basis for the automatic generation of XML templates.

3.3. XML Template Generation

After semantic understanding of protocol tables, protocol item type identification, and information extraction and integration are completed, the system enters the XML template generation stage. The inputs at this stage are no longer raw protocol texts, but two categories of structured results generated by the preceding large language model processing. One is the main-table semantic result in the form of a nine-column table header, which provides field attributes such as sequence number, information name, byte length, data type, signedness, scale factor, unit, referenced table, and remarks. The other is the protocol item type and structural description result, which describes the structural relationships and supplementary constraints of frame headers, ordinary items, bit-field parsing items, dynamic items, struct items, branch items, sub-protocol items, checksum items, and frame tails. The former provides explicit field-level semantics, while the latter provides structural semantics of protocol organization. Together, they constitute the basis for automatic XML generation.

In the specific implementation, the XML generation module first parses the above two types of input separately to construct the internal abstract representation of the proto-

col and the mapping relationship of fields in the main table. It then uses the main-table sequence number as the index to match field attributes with their corresponding protocol item types and generates node structures according to different XML mapping rules for different types. For frame headers and frame tails, the system generates nodes describing fixed or variable values. For ordinary items, the processing function is further used to distinguish among raw, physical value, BCD encoding, ASCII encoding, bit-field parsing, and enumeration forms. For dynamic items, struct items, branch items, and sub-protocol items, nested nodes are generated according to their internal element boundaries and hierarchical relations. For checksum items, the related information is written into protocol-level attributes to describe the checksum method, starting position, and checksum scope. Finally, the program outputs XML files conforming to the digital mapping model specification, thereby realizing the automatic conversion from structured LLM understanding results to standard protocol template files. This process ensures the consistency, standardization, and reproducibility of template generation while effectively reducing the workload and subjective errors caused by manual item-by-item template filling.

3.4. Prompt Optimization Process

Because resource interface protocol documents exhibit diverse formats, inconsistent descriptions, rich implicit semantics, and complex nested structures, it is difficult to ensure that large language models can output stable, parsable, and engineering-constrained results in different task stages merely by repeatedly adjusting prompts based on human experience. This is especially true in the multi-expert collaborative framework adopted in this paper, where different subtasks differ significantly in input form, focus, and output constraints. For example, main-table extraction focuses on field completeness, frame header identification focuses on fixed values and boundary judgment, while bit-field parsing recognition relies more on the consistency between bit ranges and enumeration semantics. Therefore, this paper introduces a task-data-oriented automatic optimization mechanism for prompt design. Through the closed-loop process of "evaluation–feedback–rewriting–re-evaluation", the prompts used by each expert module are iteratively refined, transforming the trial-and-error process that traditionally depends on human experience into a quantifiable, comparable, and reproducible optimization procedure.

Specifically, for key expert modules such as main-table extraction, frame header recognition, bit-field parsing recognition, dynamic-item recognition, branch-item recognition, and sub-protocol recognition, corresponding protocol-sample development sets are constructed. For the m-th expert module, its development set is denoted as Equation (1):

$$\mathcal{D}^{(m)} = \{(x_i^{(m)}, y_i^{(m)})\}_{i=1}^{N_m}, \tag{1}$$

where $x_i^{(m)}$ indicates the input protocol sample, $y_i^{(m)}$ indicates the manually annotated ground-truth result, and N_m indicates the data scale of the task.

In each optimization round, the system first uses the current prompt to drive the large language model to complete the corresponding task and then compares the output with the ground truth. To avoid evaluating prompt quality using only a single accuracy metric, this paper adopts a multi-dimensional composite scoring strategy to jointly assess format validity, field recognition accuracy, structural boundary completeness, and satisfaction of business rules. For a single sample, the composite score is defined as Equation (2):

$$\begin{aligned} s_i^{(m)} = \alpha S_{flmt}(\hat{y}_i^{(m)}, y_i^{(m)}) + \beta S_{field}(\hat{y}_i^{(m)}, y_i^{(m)}) + \gamma S_{struct}(\hat{y}_i^{(m)}, y_i^{(m)}) + \\ \delta S_{rule}(\hat{y}_i^{(m)}, y_i^{(m)}), \end{aligned} \tag{2}$$

where S_{flmt} indicates whether the output satisfies program-parsable format requirements, S_{field} indicates the accuracy of row-number, field-value, and label recognition, S_{struct} indicates the completeness of structural boundaries, nesting relations, and coverage ranges, and S_{rule} indicates whether the output satisfies protocol-related business constraints; α, β, γ, δ are corresponding weights.

Then, the overall score of the current prompt on the development set of the task can be expressed as Equation (3):

$$J^{(m)}(p_t^{(m)}) = \frac{1}{N_m}\sum_{i=1}^{N_m} s_i^{(m)}, \tag{3}$$

where the symbol $p_t^{(m)}$ denotes the prompt used by the m-th expert module at iteration t. This objective function is used to measure the overall performance of the current prompt on the task dataset and serves as the direct basis for prompt selection and updating.

In the optimization procedure, the system first uses a manually written initial prompt as the seed prompt, runs the corresponding expert module on the development set, and calculates the overall score. It then extracts failure modes from low-scoring samples, such as incorrect main-table identification, frame-header boundary deviation, missing bit-range coverage, incomplete dynamic-element coverage, or incorrect branch-path organization, and feeds these errors together with their scores back to the large language model. On this basis, the large language model generates multiple candidate prompts around the current prompt and reruns the task and scoring procedure on the same development set. The prompt used in the next round is selected according to Equation (4):

$$p_{t+1}^{(m)} = arg_{p\in\mathcal{P}_t^{(m)}} J^{(m)}(p), \tag{4}$$

where the symbol $p_t^{(m)}$ denotes the candidate prompt set generated in iteration t. In other words, the system always retains the prompt with the highest score on the current development set for the next round of optimization, thereby forming an adaptive rewriting mechanism based on task-effect feedback. To reduce the risk of overfitting prompts to a single development set, this paper further adopts an evaluation strategy of "development-set optimization, validation-set selection, and test-set verification". Specifically, prompts are automatically rewritten only on the development set, candidate versions are selected on the validation set, and their generalization performance is evaluated on independent test protocols. Through this process, prompts can continuously self-correct around specific task data, thereby supporting the stable operation of each expert module and the overall quality of protocol template generation.

In summary, the proposed method follows a staged and constrained construction route. Heterogeneous protocol documents are first converted into a unified Markdown representation. The LLM-based semantic understanding module then generates a normalized nine-column field-level representation. On this basis, protocol item type identification experts identify structural roles and parsing behaviors, and the information integration module combines field-level semantics with structure-level descriptions. Finally, the XML generation module converts these program-compatible intermediate results into standardized protocol templates. This staged design separates semantic understanding, structural reasoning, and template generation, thereby improving the controllability and reproducibility of the overall pipeline.

4. Experiments and Results

4.1. Experimental Setup

To verify the effectiveness of the proposed intelligent construction method for resource interface protocol templates, experiments are conducted on a protocol benchmark test set containing 20 protocol tables. This test set covers a variety of protocol item types, including frame headers, raw fields, physical value fields, BCD-coded fields, ASCII-coded fields, bit-field parsing fields, enumeration fields, string fields, dynamic items, dynamic elements, struct items, struct elements, branch items, sub-protocol items, checksum fields, and frame tails. This coverage includes both conventional protocol elements and complex structures, such as repeated fields, nested blocks, and cross-field dependencies. Therefore, the test set provides a representative preliminary basis for evaluating the proposed method under typical LVC protocol-table scenarios.

The 20 protocol samples used in the experiments constitute the final independent test set. They were not used for model training, model fine-tuning, prompt rewriting, or prompt selection. Since the base large language model was invoked through an API service without parameter updating, no training set was required for model-parameter learning. To avoid overfitting expert prompts to the final evaluation samples, prompt optimization and prompt selection were performed only on separate development and validation samples constructed from other protocol tables. These development and validation samples did not overlap with the final 20-sample benchmark test set. The final results reported in this section were obtained exclusively on the independent 20-sample test set.

Large language model configuration. In this study, the DeepSeek-R1-0528 chat model was used as the base large language model and was accessed through Alibaba Cloud Model Studio (Bailian) via a hosted API service. The model was used in its original released form, without downloading local model weights, fine-tuning, parameter updating, layer modification, encoder replacement, or task-specific retraining. Since the model was accessed through an API service, internal deployment details such as the exact checkpoint size and infrastructure configuration were not directly controlled by the authors.

The adaptation of the model to the protocol-template construction task relied on prompt design, expert task decomposition, structured output constraints, and downstream programmatic parsing rather than model-parameter modification. For all LLM-based experiments, the API provider's default decoding configuration was retained. No task-specific decoding-parameter tuning was conducted, including temperature tuning, top-p adjustment, sampling-strategy optimization, or other generation-parameter optimization. The maximum input and output lengths were constrained by the API service limits. Long protocol documents were first converted into Markdown format and then segmented according to protocol-table boundaries before being submitted to the corresponding LLM modules.

For all LLM-based experiments, including the proposed LLM-Expert Pipeline and the single-stage LLM baseline, the same DeepSeek-R1 API setting was used to ensure a fair comparison. The single-stage LLM baseline received the same protocol input but was required to directly generate the final structured result in one step. In contrast, the proposed LLM-Expert Pipeline invoked the same model through several task-specific expert modules, including semantic table understanding, frame-header identification, ordinary item recognition, bit-field parsing recognition, dynamic and struct item recognition, branch and sub-protocol recognition, checksum and frame-tail recognition, and final structured integration. The rule-based baseline did not use any LLM component and was implemented using predefined keyword matching, regular expressions, and deterministic structural parsing rules.

Table 3 summarizes the composition of the final 20-sample benchmark test set. The table shows the protocol item types and structural features covered by each sample, thereby

demonstrating that the selected samples provide broad coverage of the protocol structures investigated in this study.

Table 3. Experimental setup and dataset summary.

Sample ID	Protocol Name	Main Table Rows	Included Protocol Item Types
01	Pulse-Doppler Radar Target Information Transmission Protocol Data Unit Frame Definition	28	Frame Header; Raw; Physical Value; BCD; ASCII; String; Dynamic Item; Dynamic Element; Sub-Protocol Item
02	Aircraft Mission Frame Structure Definition	20	Frame Header; Raw; Physical Value; Enumeration; Struct Item; Struct Element; Branch Item
03	Electro-Optical Pod Communication Protocol I	25	Frame Header; Raw; Physical Value; Bit-Field Parsing; Enumeration; Checksum
04	Electro-Optical Pod Communication Protocol II	21	Frame Header; Raw; Physical Value; Bit-Field Parsing; Enumeration; Checksum
05	Target Detection Information Message	15	Frame Header; Raw; Physical Value; Struct Item; Struct Element
06	WRT Electro-Optical Information Message	29	Frame Header; Raw; Physical Value
07	Power Rail Status Packet	10	Frame Header; Physical Value; Struct Item; Struct Element
08	Servo Status Return Code	7	Frame Header; Physical Value; Bit-Field Parsing; Enumeration; Checksum
09	Multi-Gas Sensor Status Reporting Protocol Frame Definition	15	Frame Header; Physical Value; BCD; ASCII; Bit-Field Parsing; Enumeration; Checksum; Frame Tail
10	Sonar Track Batch Reporting Protocol Frame Definition	21	Frame Header; Raw; Physical Value; Bit-Field Parsing; Enumeration; Dynamic Item; Dynamic Element; Checksum; Frame Tail
11	Electro-Optical Tracking Mode Control Response Protocol	8	Frame Header; Raw; Enumeration; Branch Item; Branch Element; Checksum; Frame Tail
12	Device Inspection Record Reporting Protocol	17	Frame Header; Raw; Physical Value; BCD; ASCII; Struct Item; Struct Element; Checksum; Frame Tail
13	UAV Status Broadcast Protocol	12	Frame Header; Raw; Physical Value; Bit-Field Parsing; Checksum
14	GNSS Position and Heading Serial Text Protocol	11	Frame Header; Raw; Physical Value; Enumeration; Checksum
15	Power Supply Module Query and Control Protocol	5	Frame Header; Raw; Enumeration; Checksum
16	CAN Bus Object Access Protocol	5	Frame Header; Raw; Bit-Field Parsing
17	Electro-Optical Device Target Batch Feedback Protocol	6	Frame Header; Raw; Dynamic Item; Dynamic Element; Checksum
18	Sea Surface Buoy Event Reporting Protocol	7	Frame Header; Raw; Enumeration; Sub-Protocol Item; Checksum
19	Vehicle Terminal Device Inspection Result Protocol	7	Frame Header; BCD; ASCII; Bit-Field Parsing; Enumeration; Frame Tail
20	Electromechanical Equipment Maintenance Record Upload Protocol	6	Frame Header; ASCII; String; Dynamic Item; Dynamic Element; Checksum

To ensure fair comparison and interpretable analysis, the experiments were designed at two complementary evaluation levels. The first level is the protocol understanding and structured extraction level, which mainly evaluates the ability of different methods to identify and extract the main protocol table and determine protocol item types. The second level is the end-to-end XML template generation level, which mainly evaluates whether the generated intermediate results can be successfully parsed by the XML generation program and how consistent the generated XML is with the gold-standard template at the node level. The reason for adopting this layered design is that XML generation in this work relies on strict intermediate representations. Therefore, it is necessary to separately evaluate protocol semantic understanding capability and system-level deployability.

At the protocol understanding and structured extraction level, three methods were compared: a rule-based baseline (rule_based), a single-stage large language model baseline (single_stage_llm), and the proposed method (LLM-Expert Pipeline). At the XML template generation level and in the ablation study, three degraded variants were further considered: removing the semantic understanding module (w/o semantic_understanding), removing the type expert module (w/o type_experts), and a single-stage generation method (single_stage_generation). This design makes it possible to verify both the overall advantage of the proposed method over the baselines and the specific contribution of each key module to system performance.

Research questions and evaluation metrics: The evaluation metrics were selected according to the main research questions of this study. Since the proposed method aims to convert heterogeneous protocol documents into executable XML protocol templates, the evaluation should cover not only protocol-table understanding but also protocol item type recognition and downstream XML-generation reliability. Therefore, four research questions were defined, as summarized in Table 4.

Table 4. Relationship between research questions and evaluation metrics.

Research Question	Evaluation Aspect	Metrics	Rationale
RQ1: Can the method correctly identify and normalize the main protocol table?	Protocol table understanding and semantic normalization	Main-table cell accuracy; row-level exact match	These metrics evaluate whether field-level attributes and complete minimum field units are correctly extracted into the nine-column representation.
RQ2: Can the method correctly recognize protocol item types and structural relationships?	Protocol item type recognition and structure modeling	Type-recognition F1-score; macro-F1-score	These metrics evaluate both overall classification accuracy and balanced performance across frequent and infrequent protocol item categories.
RQ3: Can the structured outputs be converted into XML templates?	XML-generation feasibility	Parser-compatible rate; XML generation success rate	These metrics evaluate whether intermediate outputs satisfy downstream input constraints and whether XML files can be generated successfully.
RQ4: Are the generated XML templates structurally correct and stable?	XML structural correctness and engineering reliability	Node consistency; runtime error rate	These metrics evaluate node-level agreement with gold-standard XML and practical execution stability.

Specifically, main-table cell accuracy measures whether the field-level attributes in the normalized nine-column table are correctly extracted, while row-level exact match measures whether an entire minimum field unit is completely consistent with the reference result. Type-recognition F1-score evaluates the overall correctness of protocol item classification, and macro-F1 gives equal weight to different protocol item categories, which is useful for evaluating less frequent but structurally important categories. Parser-compatible rate measures whether the generated intermediate results satisfy the input constraints of the XML generation program, whereas XML generation success rate evaluates whether XML files can be successfully generated. Node consistency measures the structural consistency between the predicted XML and the gold-standard XML at the node level, and runtime error rate reflects engineering stability during practical execution. In this way, the selected metrics jointly evaluate semantic correctness, type-level reasoning accuracy, template-generation feasibility, and engineering reliability.

It should be noted that a value of 1.0 for parser-compatible rate or XML generation success rate indicates that all tested samples can be parsed by the downstream XML generation program or successfully converted into XML files. These metrics mainly measure execution feasibility and generation completeness. They do not imply perfect XML structural correctness. Structural correctness is evaluated separately using node consistency, which measures the node-level agreement between the generated XML and the gold-standard XML template.

4.2. Results on Protocol Understanding and Structured Extraction

Table 5 summarizes the overall performance of the three methods at the protocol understanding and structured extraction level, while Figure 6 provides a visual comparison of the major evaluation metrics. As can be observed, the proposed LLM-Expert Pipeline achieves the best performance on all core indicators, demonstrating a clear overall advantage over the comparison methods. Specifically, the proposed method attains 0.9761 in main-table extraction accuracy, 0.8062 in row-level exact match, 0.9769 in type-recognition F1-score, and 0.9824 in macro-F1. In contrast, the rule-based method achieves only 0.7462 and 0.4551 in main-table extraction accuracy and type-recognition F1-score, respectively, while the corresponding values for the single-stage LLM method are 0.6564 and 0.5155. These results indicate that decomposing protocol parsing into a staged pipeline of semantic understanding, type identification, and structural integration can substantially improve both the accuracy and stability of protocol table analysis.

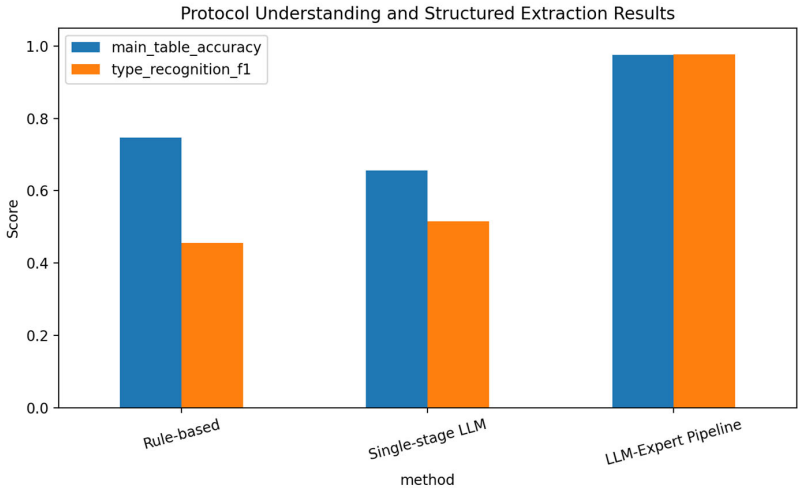


Figure 6. Overall comparison of protocol understanding and structured extraction performance.

Table 5. Overall results on protocol understanding and structured extraction.

Method	Main-Table Accuracy	Row-Level Exact Match	Type Recognition F1-Score	Macro-F1
Rule-Based	0.7462	0.1471	0.4551	0.7846
Single-Stage LLM	0.6564	0.1967	0.5155	0.7667
LLM-Expert Pipeline	0.9761	0.8062	0.9769	0.9824

From the perspective of main-table extraction, the rule-based method still performs reasonably well on samples with relatively regular layouts and fixed field descriptions. However, its performance degrades markedly when auxiliary tables, remarks, and cross-table semantic relations are involved. Although the single-stage LLM method has stronger semantic modeling capability, its outputs still exhibit considerable fluctuations in main-table boundary identification, minimum-unit splitting, and field-level consistency, mainly due to the lack of explicit task decomposition and structural constraints. As a result, although its row-level exact match is slightly higher than that of the rule-based method, its overall main-table extraction accuracy remains unsatisfactory. By contrast, the proposed method first explicitly identifies the main table and its associated supplementary semantics through a dedicated semantic understanding module and then outputs standardized results under a unified nine-column schema. This design enables the method to maintain higher consistency and completeness in complex protocol scenarios. The trend shown in Figure 6 further supports this observation: the proposed method not only leads in main-table extraction accuracy, but also maintains consistently superior performance in type recognition.

To further examine the performance differences across protocol item categories, Table 6 reports the precision, recall, and F1-score for each fine-grained protocol item type, and Figure 7 presents a visual comparison of the F1-scores across categories. Overall, the proposed method achieves near-perfect recognition performance on most protocol item types. In particular, the F1-scores for Frame Header, String, Dynamic Item, Struct Item, Struct Element, Branch Item, Sub-Protocol Item, Checksum, and Frame Tail reach 1.0000 or remain very close to 1.0000. Even for more challenging categories, including Raw, Physical Value, BCD, ASCII, Bit-Field Parsing, and Enumeration, the proposed method still achieves high F1-scores of 0.9485, 0.9784, 0.9565, 0.9474, 0.9730, and 0.9714, respectively. These results suggest that the expert-guided type identification and structural constraint mechanism is effective not only for common field types, but also for more complex protocol items involving bit-level semantics, value mapping, and dynamic structure boundary inference.

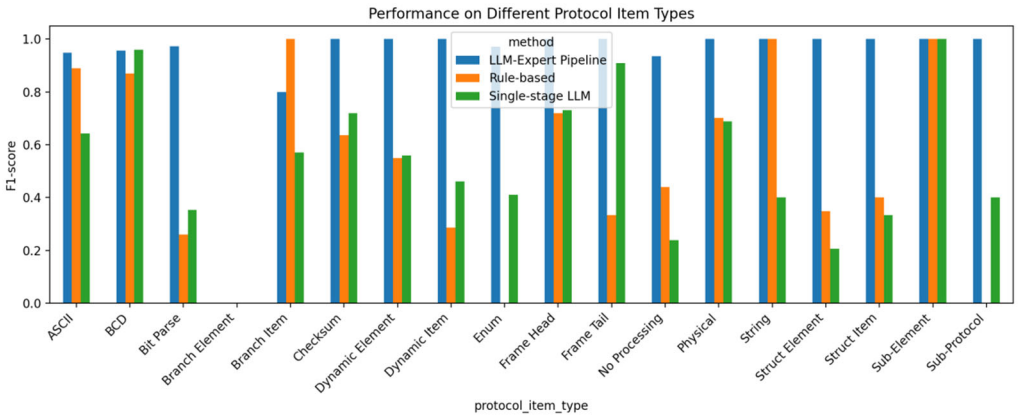


Figure 7. F1-scores on different protocol item types.

Table 6. Results on different protocol item types.

Method	Protocol Item Type	Precision	Recall	F1-Score
Rule-Based	Frame Header	0.9697	0.5714	0.7191
Rule-Based	Raw	0.3934	0.5	0.4404
Rule-Based	Physical Value	0.8542	0.5942	0.7009
Rule-Based	BCD	0.9091	0.8333	0.8696
Rule-Based	ASCII	1	0.75	0.8571
Rule-Based	Bit-Field Parsing	0.2308	0.3	0.2609
Rule-Based	Enumeration	0	0	0
Rule-Based	String	0	0	0
Rule-Based	Dynamic Item	1	0.1667	0.2857
Rule-Based	Dynamic Element	0.7	0.4516	0.549
Rule-Based	Struct Item	1	1	1
Rule-Based	Struct Element	1	0.9565	0.9778
Rule-Based	Branch Item	1	0.5	0.6667
Rule-Based	Branch Element	0	0	0
Rule-Based	Sub-Protocol Item	1	0.5	0.6667
Rule-Based	Sub-Protocol Element	0	0	0
Rule-Based	Checksum	0.9091	0.6667	0.7692
Rule-Based	Frame Tail	1	0.6667	0.8
Single-Stage LLM	Frame Header	1	0.6429	0.7826
Single-Stage LLM	Raw	0.5	0.2292	0.3143
Single-Stage LLM	Physical Value	0.725	0.8406	0.7786
Single-Stage LLM	BCD	1	0.8333	0.9091
Single-Stage LLM	ASCII	1	0.875	0.9333
Single-Stage LLM	Bit-Field Parsing	0.1818	0.2	0.1905
Single-Stage LLM	Enumeration	0.1818	0.1429	0.16
Single-Stage LLM	String	1	0.5	0.6667
Single-Stage LLM	Dynamic Item	0.8	0.6667	0.7273
Single-Stage LLM	Dynamic Element	0.6957	0.5161	0.5926
Single-Stage LLM	Struct Item	0.8333	1	0.9091
Single-Stage LLM	Struct Element	0.8182	0.7826	0.8
Single-Stage LLM	Branch Item	1	0.5	0.6667
Single-Stage LLM	Branch Element	0	0	0
Single-Stage LLM	Sub-Protocol Item	1	0.5	0.6667
Single-Stage LLM	Sub-Protocol Element	0	0	0
Single-Stage LLM	Checksum	0.9091	0.8333	0.8696
Single-Stage LLM	Frame Tail	0.75	0.5	0.6
LLM-Expert Pipeline	Frame Header	1	1	1
LLM-Expert Pipeline	Raw	0.9388	0.9583	0.9485
LLM-Expert Pipeline	Physical Value	0.9714	0.9855	0.9784
LLM-Expert Pipeline	BCD	0.9167	1	0.9565
LLM-Expert Pipeline	ASCII	0.9	1	0.9474
LLM-Expert Pipeline	Bit-Field Parsing	0.9474	1	0.973
LLM-Expert Pipeline	Enumeration	1	0.9444	0.9714
LLM-Expert Pipeline	String	1	1	1
LLM-Expert Pipeline	Dynamic Item	1	1	1
LLM-Expert Pipeline	Dynamic Element	0.9688	1	0.9841
LLM-Expert Pipeline	Struct Item	1	1	1
LLM-Expert Pipeline	Struct Element	1	1	1
LLM-Expert Pipeline	Branch Item	1	1	1
LLM-Expert Pipeline	Branch Element	0	0	0
LLM-Expert Pipeline	Sub-Protocol Item	1	1	1
LLM-Expert Pipeline	Sub-Protocol Element	0	0	0
LLM-Expert Pipeline	Checksum	1	1	1
LLM-Expert Pipeline	Frame Tail	1	1	1

In contrast, the rule-based method performs poorly on several categories that involve complex structures or strong dependence on contextual semantics. For example, its F1-scores on Bit-Field Parsing and Enumeration are only 0.2609 and 0.0000, respectively, indicating that fixed rules are insufficient for identifying protocol items whose interpretation depends on auxiliary tables, bit-definition descriptions, or remark texts. A similar limitation can be observed for Dynamic Item and Dynamic Element, where the rule-based method achieves F1-scores of only 0.2857 and 0.5490, respectively, revealing its weakness in recognizing dynamically repeated structures. The single-stage LLM method attains relatively competitive results on several conventional field types, such as Physical Value, BCD, and ASCII, but remains unstable on categories that require explicit structural modeling or cross-context integration, including Raw, Bit-Field Parsing, Enumeration, Struct Element, and Sub-Protocol Item. This tendency is particularly evident in Figure 7, where the single-stage method exhibits larger performance fluctuations across categories, whereas the proposed method shows a much more stable and uniformly high distribution.

It should also be noted that the number of samples for a few complex categories remains limited in the current test set. For example, the support for Branch Element is only 1. Under such conditions, a temporarily low F1-score for a particular class should not be directly interpreted as evidence that the method is ineffective for that structure; instead, it is more likely to reflect the high variance of evaluation results under extremely small sample sizes. Overall, the results in Tables 5 and 6, as well as the trends in Figures 6 and 7, show that the proposed method improves both main-table extraction and fine-grained item recognition. These improvements provide more reliable structured inputs for XML template generation.

4.3. XML-Level Evaluation of Template Generation

On the basis of the performance achieved at the protocol understanding and structured extraction level, this paper further evaluates the three methods at the end-to-end XML template generation level. Table 7 reports the overall XML-level results, while Figure 8 compares the parser compatibility, XML success rate, and node consistency of the three methods. As shown in Table 7, the LLM-Expert Pipeline achieves the best results on all four XML-level metrics, with a parser-compatible rate of 1.0, an XML generation success rate of 1.0, a node consistency of 0.9478, and a runtime error rate of 0. In contrast, the rule-based method achieves an XML generation success rate of 0.95 and a node consistency of only 0.5376, while the single-stage large language model method further drops to an XML generation success rate of 0.70 and a node consistency of only 0.4268, with a runtime error rate as high as 0.30.

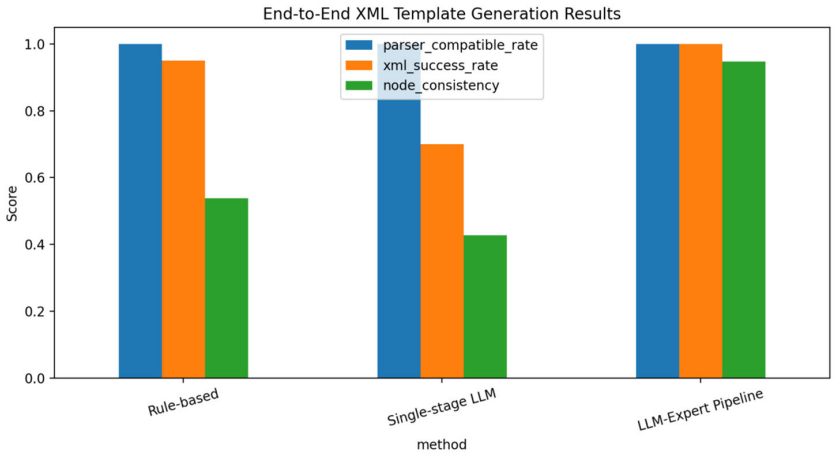


Figure 8. Comparison of parser compatibility, XML success rate, and node consistency.

Table 7. End-to-end XML template generation results.

Method	Parser Compatibility Rate	XML Generation Success Rate	Node Consistency	Runtime Error Rate
Rule-Based	1	0.95	0.5376	0.05
Single-Stage LLM	1	0.70	0.4268	0.30
LLM-Expert Pipeline	1	1.00	0.9478	0

The values of 1.0 for parser-compatible rate and XML generation success rate should be interpreted as evidence of execution feasibility and program compatibility of the end-to-end workflow. In the proposed pipeline, the outputs of the large language model are first constrained into structured intermediate representations, and the final XML files are generated by deterministic programmatic procedures. Therefore, once the intermediate results satisfy the required input format and contain the necessary structural fields, the XML generation program can successfully produce XML files for all test samples.

However, this does not imply that the generated XML templates are completely correct. The correctness of the XML content still depends on whether the upstream LLM-based modules correctly identify field semantics, protocol item types, structural boundaries, dynamic groups, sub-protocol references, enumeration mappings, and checksum information. Therefore, XML structural correctness is evaluated separately using node consistency. The node consistency of 0.9478 indicates that the generated XML templates are highly consistent with the gold-standard templates, while small structural deviations still remain in some complex cases. These results show that the proposed method can not only improve the structured representation of protocol semantics, but also produce program-compatible intermediate results that support stable XML template generation.

It is worth noting that although the rule-based method can generate formally valid XML files for most samples, its node consistency is obviously low, suggesting that its XML outputs are more "generatable" than "structurally correct". The situation is even more severe for the single-stage method. It not only yields the lowest node consistency, but also suffers from significant runtime anomalies. Sample-wise results show that single_stage_llm produces errors such as 'list' object has no attribute 'items' on multiple samples, causing XML generation to fail directly. In a few samples, abnormal scalar values such as 7 are even propagated to downstream procedures, further indicating serious drift in the structured format of single-stage outputs. In other words, the problem of the single-stage method is not merely reduced accuracy, but rather the lack of sufficient engineering stability in the intermediate results themselves.

By contrast, the full method explicitly preserves the three stages of semantic understanding, type recognition, and structural integration, enabling the XML generation program to receive more stable inputs and avoiding the problem that "part of the semantics may be understood, but the intermediate representation cannot be reliably deployed". Therefore, the XML-level experiments verify the second major advantage of the proposed method, namely, that it produces more reliable structured intermediate representations and better fits the automated workflow for digital mapping model construction.

4.4. Ablation Study

To analyze the specific contribution of each key module to system performance, an ablation study is further conducted. Table 8 presents the overall results of the complete method and its degraded variants. The full LLM-Expert Pipeline achieves the best performance at both the understanding level and the XML level, with a main-table extraction accuracy of 0.9761, a type-recognition F1-score of 0.9769, an XML generation success rate of 1.0, and a node consistency of 0.9478. When the semantic understanding module is

removed, the main-table extraction accuracy drops to 0.7462, the type-recognition F1-score decreases to 0.4551, and node consistency declines to 0.5376. When the type expert module is removed, the main-table extraction accuracy remains at 0.9761, but the type-recognition F1-score also drops to 0.4551, and node consistency is only 0.5698. The single-stage generation method performs the worst overall, with a main-table extraction accuracy of only 0.6564, a type-recognition F1-score of 0.5155, an XML success rate of 0.7, a node consistency of 0.4268, and a runtime error rate of 0.3.

Table 8. Ablation results of the proposed pipeline.

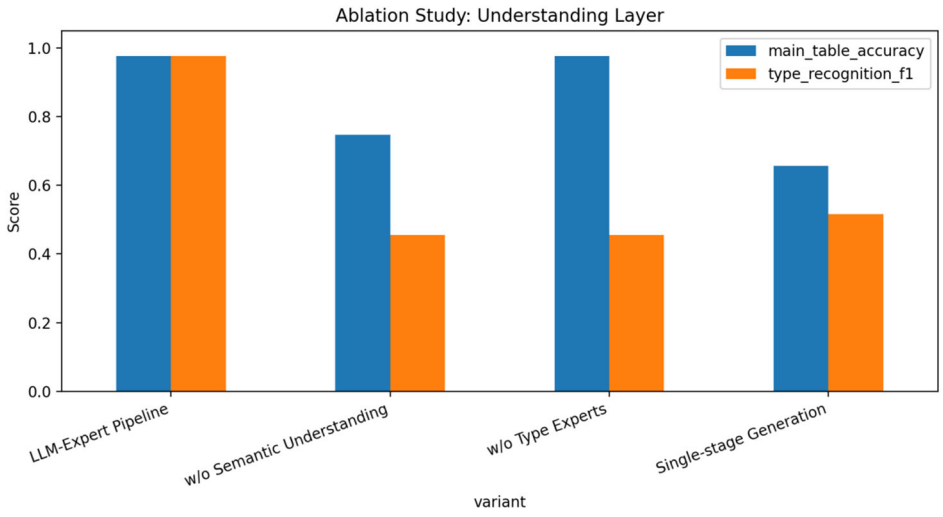
Variant	Main-Table Accuracy	Type-Recognition F1	Parser-Compatible Rate	XML Success Rate	Node Consistency	Runtime Error Rate
LLM-Expert Pipeline	0.9761	0.9769	1	1	0.9478	0
w/o semantic understanding	0.7462	0.4551	1	0.95	0.5376	0.05
w/o type experts	0.9761	0.4551	1	0.95	0.5698	0.05
Single-stage generation	0.6564	0.5155	1	0.7	0.4268	0.3

These results, as shown in Figure 9, clearly reveal the role of each module. The semantic understanding module mainly contributes to main-table identification, minimum-unit splitting, and auxiliary-table association. These operations provide unified and stable inputs for subsequent protocol item modeling. Therefore, once this module is removed, both main-table extraction performance and XML node consistency decline significantly. The type expert module mainly contributes to the modeling of complex protocol item structures, especially those requiring structural reasoning, such as bit-field parsing, dynamic items, struct items, and sub-protocol items. After removing the type expert module, the main table can still be preserved relatively well, but the type-recognition F1-score drops sharply, indicating that the expert module is crucial for complex protocol structure modeling. Finally, the poor performance of the single-stage generation method further indicates that compressing protocol understanding, type recognition, and structured output into a single generation step significantly harms output stability and system usability.

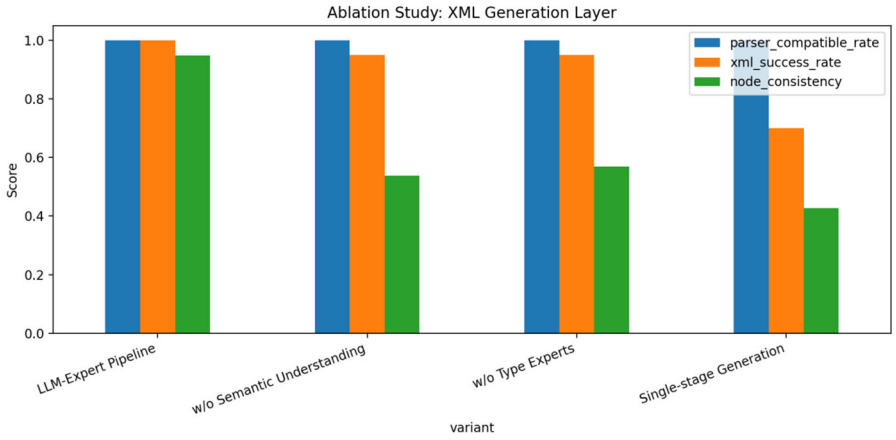
From the XML-level perspective, although all three ablation variants still maintain a parser-compatible rate of 1.0, node consistency and runtime error rate can still effectively distinguish between "runnable" and "reliably runnable". This indicates that merely passing the input checks of the XML generator is far from sufficient to demonstrate method effectiveness; node-level structural correctness and engineering stability remain the key criteria for evaluating resource interface protocol template construction methods.

4.5. Prompt Sensitivity, Transferability, and Cost Analysis

Since the proposed LLM-Expert Pipeline relies on expert prompt templates and prompt optimization, this subsection further analyzes the influence of prompt design on system performance, transferability, and engineering cost. This analysis clarifies the source of the performance improvement. It examines whether the improvement comes from the base LLM alone or from the combined use of task-specific prompt decomposition, structured output constraints, and downstream programmatic validation.



(**a**)



(**b**)

Figure 9. (**a**). Ablation results on protocol understanding. (**b**). Ablation results on XML generation.

First, during prompt development, we qualitatively compared the behavior of manually designed initial prompts, optimized expert prompts, and simplified prompts with reduced structural constraints. The initial prompts contained only the basic task definition and output requirements. The optimized prompts were obtained through the evaluation–feedback–rewriting process described in Section 3.4. The simplified prompts retained the same task goal but removed part of the explicit evidence requirements, boundary constraints, and output-format constraints. The comparison showed that weaker structural constraints caused more unstable outputs. Typical problems included incomplete row-number sets, unstable dynamic-item boundaries, missing auxiliary-table associations, and intermediate outputs that were difficult to parse. In contrast, the optimized expert prompts improved output consistency mainly by reducing format errors and structural-boundary deviations.

Second, prompt transferability was analyzed at the protocol-item type level. The same expert prompt templates were applied to protocol-table samples involving different resource types and structural patterns, including sensor status packets, electro-optical device messages, serial text protocols, CAN bus object access protocols, and maintenance-record upload protocols. The analysis indicates that prompts describing general protocol-processing principles, such as frame-header identification, physical-value recognition, BCD and ASCII recognition, enumeration recognition, and checksum identification, are relatively transferable across protocol-table samples. However, prompts for dynamic items, branch

items, and nested sub-protocols are more sensitive to document style, table layout, remarks, and auxiliary-table organization, because these structures depend strongly on cross-row and cross-table semantic evidence.

Third, the cost of prompt tuning was considered together with the performance gain. Compared with the single-stage LLM baseline, the proposed method requires multiple LLM invocations because different expert modules are called separately. This increases the engineering overhead of prompt design, prompt-version management, and API usage. However, prompt optimization is performed mainly during the development stage; after the expert prompts are finalized, the inference process uses fixed prompts and deterministic downstream XML-generation procedures. In addition, the ablation results show that removing the semantic understanding module or the type expert module leads to clear decreases in type-recognition F1-score and XML node consistency, while the single-stage generation method suffers from lower XML generation success and higher runtime error rates. Therefore, the additional prompt-engineering cost is mainly exchanged for higher structural stability, better program compatibility, and more reliable XML-template generation.

Table 9 summarizes the main observations obtained from the prompt sensitivity, transferability, and cost analysis. Overall, prompt design has a non-negligible influence on the proposed pipeline. The prompts are relatively transferable for common field types, but more style-sensitive for complex structures such as dynamic items, branch items, and sub-protocols. This also indicates that future work should further investigate automatic prompt validation, prompt version control, and lightweight prompt adaptation strategies for new protocol domains.

Table 9. Qualitative analysis of prompt sensitivity, transferability, and engineering cost.

Analysis Aspect	Observation	Main Implication
Prompt sensitivity	Simplified prompts were more likely to produce incomplete row-number sets, unstable structural boundaries, missing auxiliary-table associations, or unparsable intermediate outputs.	Explicit evidence requirements, boundary constraints, and output-format constraints are important for XML-generation reliability.
Transferability of common-field prompts	Prompts for frame headers, raw fields, physical values, BCD, ASCII, enumerations, and checksum items were relatively transferable across protocol-table samples.	Common protocol-processing prompts can be reused with limited adaptation.
Transferability of complex-structure prompts	Prompts for dynamic items, branch items, and sub-protocols were more sensitive to table layout, remarks, and auxiliary-table organization.	Complex structural prompts may require domain-specific adaptation.
Prompt-tuning cost	The pipeline requires multiple LLM invocations, prompt design, and prompt-version management.	Engineering overhead is higher than single-stage generation.
Performance gain	Ablation results show improvements in type-recognition F1-score, XML success rate, node consistency, and runtime stability.	The additional prompt-engineering cost is justified when structural correctness and program compatibility are required.

4.6. Illustrative Case Study of Protocol Template Construction

To further clarify the structure and use of the proposed protocol template, this subsection presents an illustrative case study corresponding to the staged workflow described in the system architecture and XML generation modules. To avoid dependence on any project-specific protocol format, a representative generic protocol table is constructed as an example, as shown in Tables 10 and 11. The case follows the complete processing route from an original protocol table to the standardized nine-column semantic representation,

then to the protocol item type and structural description result, and finally to the generated XML protocol template. For readability, this example is referred to as the Generic Target Status Reporting Protocol.

Table 10. Original protocol table excerpt for the Target Information Reporting Protocol.

No.	Information Name	Byte Length	Data Type	Quantization Unit	Description
1	Message sequence number	1	Unsigned integer		The value is set during packet generation.
2	Message identifier	1	Unsigned integer		A0H.
3	Message length	2	Unsigned integer		--
4	ASCII status identifier	1	Unsigned integer		ASCII-coded identifier.
5	ASCII timestamp identifier	1	Unsigned integer		ASCII-coded identifier.
6	ASCII sender identifier	1	Unsigned integer		ASCII-coded identifier.
7	Second	1	Unsigned integer		BCD code, range: 0–59.
8	Minute	1	Unsigned integer		BCD code, range: 0–59.
9	Hour	1	Unsigned integer		BCD code, range: 0–23.
10	Position-information message type	1	Unsigned integer		See Protocol Auxiliary Table 1.
11	Type indicator	1	Unsigned integer		0x00: Type A; 0x01: Type B; 0x02: Type C; 0x03: Type D; 0x04: Type E.
12	Device ID	4	String		String such as V1.0-XXXX; the prefix indicates the software version and the suffix indicates the hardware identifier.
13	Target count N	1	Unsigned integer		Indicates the number of targets contained in the current frame.
14	Target flag	2	Unsigned integer		Status flag used to identify target attributes.
15	Target ID	2	Unsigned integer		Unique identifier used to distinguish different targets.
16	Target status	2	Unsigned integer		Describes the current target state.
17	Coordinate	4	Unsigned integer	0.01 m	Three-dimensional position coordinate; the raw value is multiplied by 0.01.
18	X-direction velocity	4	Signed integer	0.1 m/s	Velocity in the X direction; the sign indicates direction.
19	Y-direction velocity	4	Signed integer	0.1 m/s	Velocity in the Y direction; the sign indicates direction.
20	Z-direction velocity	4	Signed integer	0.1 m/s	Velocity in the Z direction; the sign indicates direction.
21	…				Omitted repeated target records.
22–28	Repeated target record	2/4	Unsigned or signed integer	0.01 m; 0.1 m/s	The target record fields are repeated according to the target count N.
29	Checksum word	1	Unsigned integer		Checksum = XOR over bytes 3–29.

Note: The symbol "--" indicates that no additional description is provided for the field; the ellipsis "…" indicates that repeated target-record fields are omitted for compact presentation.

This example was selected because it provides a compact but complete processing path that covers the main protocol item types discussed in this paper, including frame headers, raw items, BCD items, bit-field parsing items, enumeration items, string items, dynamic items and dynamic elements, sub-protocol items, physical-value items, and checksum items. Therefore, the case is useful not only for showing the appearance of the proposed template,

but also for explaining how the semantic representation and the structural description jointly support XML generation.

Table 11. Protocol Auxiliary Table 1.

No.	Information Name	Byte Length	Data Type	Quantization Unit	Description
1	Message sequence number	1	Unsigned integer		The value is set during packet generation.
2	Message identifier	1	Unsigned integer		A1H.
3	Message length	2	Unsigned integer		0015H.
4	Longitude	4	Unsigned integer	0.000001 degree	Real value = raw value × 0.000001.
5	Latitude	4	Unsigned integer	0.000001 degree	Real value = raw value × 0.000001.
6	Altitude	2	Unsigned integer	0.1 m	Real value = raw value × 0.1.

The explicit reference in row 10 is important because it shows how auxiliary-table association is handled. During semantic understanding, the main table does not simply store the phrase 'See Protocol Auxiliary Table 1'; instead, the referenced table is retained and later converted into a nested sub-protocol structure. This avoids losing cross-table semantics during XML generation.

After main-table identification, minimum-unit splitting, and auxiliary-table association, the original table is normalized into the fixed nine-column semantic representation shown in Table 12. This table provides field-level attributes, while preserving references and remarks that are needed by the subsequent expert modules.

Table 12. Nine-column semantic representation generated from the example protocol table.

No.	Information Name	Byte Length	Data Type	Signed/Unsigned	Scale Factor	Unit	Referenced Table	Remarks
1	Message sequence number	1	unsigned integer	unsigned			Protocol Auxiliary-Table 1	Value set during packet generation.
2	Message identifier	1	unsigned integer	unsigned			Protocol Auxiliary-Table 1	Fixed value: A0H.
3	Message length	2	unsigned integer	unsigned			Protocol Auxiliary-Table 1	Protocol length field.
4	ASCII status identifier	1	uint8_t	unsigned				ASCII-coded field.
5	ASCII timestamp identifier	1	uint8_t	unsigned				ASCII-coded field.
6	ASCII sender identifier	1	uint8_t	unsigned				ASCII-coded field.
7	Second	1	uint8_t	unsigned				BCD code; range: 0–59.
8	Minute	1	uint8_t	unsigned				BCD code; range: 0–59.
9	Hour	1	uint8_t	unsigned				BCD code; range: 0–23.
10	Position-information message type	1	uint8_t	unsigned			Protocol Auxiliary-Table 1	Sub-protocol item; the referenced table has its own frame header and elements.
11	Type indicator	1	uint8_t	unsigned				0x00 Type A; 0x01 Type B; 0x02 Type C; 0x03 Type D; 0x04 Type E.
12	Device ID	4	string					String item; example format: V1.0-XXXX.
13	Target count N	1	uint8_t	unsigned				Dynamic controller; indicates the number of target records.
14	Target flag	2	uint16_t	unsigned				Dynamic element; repeated according to N.
15	Target ID	2	uint16_t	unsigned				Dynamic element; repeated according to N.
16	Target status	2	uint16_t	unsigned				Dynamic element; repeated according to N.
17	Coordinate	4	uint32_t	unsigned	0.01	m		Physical-value dynamic element.
18	X-direction velocity	4	int32_t	signed	0.1	m/s		Physical-value dynamic element.
19	Y-direction velocity	4	int32_t	signed	0.1	m/s		Physical-value dynamic element.

Table 12. *Cont.*

No.	Information Name	Byte Length	Data Type	Signed/Unsigned	Scale Factor	Unit	Referenced Table	Remarks
20	Z-direction velocity	4	int32_t	signed	0.1	m/s		Physical-value dynamic element.
21	...							Placeholder indicating omitted repeated records; not mapped as a data node.
22	Target flag	2	uint16_t	unsigned				Second occurrence of the repeated target record.
23	Target ID	2	uint16_t	unsigned				Second occurrence of the repeated target record.
24	Target status	2	uint16_t	unsigned				Second occurrence of the repeated target record.
25	Coordinate	4	uint32_t	unsigned	0.01	m		Second occurrence of the physical-value dynamic element.
26	X-direction velocity	4	int32_t	signed	0.1	m/s		Second occurrence of the physical-value dynamic element.
27	Y-direction velocity	4	int32_t	signed	0.1	m/s		Second occurrence of the physical-value dynamic element.
28	Z-direction velocity	4	int32_t	signed	0.1	m/s		Second occurrence of the physical-value dynamic element.
29	Checksum word	1	unsigned integer	unsigned				Checksum = XOR over bytes 3–29.

After the nine-column semantic representation is obtained, the protocol item type identification expert modules generate a second structured intermediate result, namely the protocol item type and structural description result. Different from the nine-column table, which mainly records field-level attributes such as field names, byte lengths, data types, units, scale factors, references, and remarks, this result describes the organizational structure of the protocol frame. It records the row-number sets corresponding to different protocol item types and further specifies structural relationships that cannot be fully expressed by a flat table, including dynamic-element grouping, nested sub-protocol header/content/tail boundaries, enumeration mappings, checksum parameters, and frame-tail information. Therefore, this result serves as a program-parsable bridge between semantic understanding and XML template generation. Listing 1 shows the formatted structural description generated for the example. In this listing, the numbers enclosed in square brackets denote row-number sets in the protocol table rather than literature citations.

Listing 1. Protocol item type and structural description result generated for the example.

```
FrameHeader: [1,2,3]
1: null
2: A0H
3: null

RawItem: [21]

PhysicalValue: [17,18,19,20,25,26,27,28]

BCDItem: [7,8,9]

ASCIIItem: [4,5,6]

BitFieldParsingItem: null

EnumerationItem: [11]
```

Listing 1. *Cont.*

11:
0x00: Type A
0x01: Type B
0x02: Type C
0x03: Type D
0x04: Type E

StringItem: [12]

DynamicItem: [13]
DynamicElements: [14,15,16,17,18,19,20;22,23,24,25,26,27,28]

StructItem: null
StructElements: null

BranchItem: null
BranchElements: null

SubProtocolItem: [10]
SubProtocolElements: []

A: Position Information Message Type Protocol

SubProtocolHeader:
No.	Information Name	Byte Length	Data Type	Signed/Unsigned	Scale Factor	Unit	Referenced Table	Remarks
A1	Message sequence number	1	unsigned integer	unsigned integer				The value is set during packet packaging
A2	Message identifier	1	unsigned integer	unsigned integer				A1H
A3	Message length	2	unsigned integer	unsigned integer				0015H

SubProtocolContent:
No.	Information Name	Byte Length	Data Type	Signed/Unsigned	Scale Factor	Unit	Referenced Table	Remarks
A4	Longitude	4	unsigned integer	unsigned integer	0.000001	degree		Physical value = raw value × 0.000001
A5	Latitude	4	unsigned integer	unsigned integer	0.000001	degree		Physical value = raw value × 0.000001
A6	Altitude	2	unsigned integer	unsigned integer	0.1	m		Physical value = raw value × 0.1

SubProtocolTail:
No.	Information Name	Byte Length	Data Type	Signed/Unsigned	Scale Factor	Unit	Referenced Table	Remarks

RelatedSupplementaryTables:

Listing 1. *Cont.*

```
null

ChecksumItem: [29]
ChecksumMethod: CheckSum_8bit
ChecksumStartPosition: 3
ChecksumRange: 27

FrameTail: null
```

In Listing 1, rows 1–3 form the frame header. Rows 4–6 correspond to ASCII fields, rows 7–9 to BCD fields, row 11 to an enumeration field, and row 12 to a string field. Row 13 controls the number of target records, so rows 14–20 and 22–28 are grouped as repeated dynamic elements. In the XML template, these rows are abstracted into one repeated structure controlled by the target-count field. Row 10 points to a auxiliary table and is therefore converted into a nested sub-protocol with a header, content section, and empty tail. Row 29 provides the checksum information, including the method, start position, and range.

Before XML generation, the formatted structural description is integrated with the nine-column semantic representation. The nine-column table provides field-level attributes, including field names, byte lengths, data types, units, scale factors, references, and remarks, whereas the structural description result provides the organizational information required for XML construction, such as frame-header grouping, dynamic-element grouping, nested sub-protocol boundaries, enumeration mappings, checksum attributes, and frame-tail information. Together, these two intermediate results enable the XML generation module to construct frame-header nodes, ordinary element nodes, repeated dynamic structures, nested protocol nodes, enumeration definitions, and protocol-level checksum attributes.

Finally, the XML generation module constructs the protocol template by integrating the nine-column semantic representation with the protocol item type and structural description result. To make the mapping process clearer, Listing 2 presents representative XML nodes grouped according to the protocol item types identified in the previous step. This presentation highlights how frame headers, ordinary decoded fields, nested sub-protocols, enumeration definitions, string fields, dynamic repeated structures, and checksum attributes are mapped into the final XML template.

Listing 2. Excerpt of the generated XML protocol template.

```
<ProtocolInfo>
<ProtocolItem name = “Target Information Reporting Protocol”
type = “ordinaryProtocol”
hasChecksum = “yes”
checksumMethod = “CheckSum_8bit”
checksumStartPosition = “3”
checksumRange = “27”>
<FrameHeaderInfo count = “3”>
<FrameHeader order = “1” name = “Message sequence number”
variable = “yes” dataType = “uint8_t” value = “0” />
<FrameHeader order = “2” name = “Message identifier”
variable = “no” dataType = “uint8_t” value = “A0H” />
<FrameHeader order = “3” name = “Message length”
```

Listing 2. *Cont.*

```
variable = “yes” dataType = “uint16_t” value = “0” />
</FrameHeaderInfo>

<ElementInfo count = “10”>
<Element order = “1” name = “ASCII status identifier”
itemType = “ordinaryItem” processingFunction = “ASCII” />
<Element order = “4” name = “Second”
itemType = “ordinaryItem” processingFunction = “BCD”
remarks = “BCD code, range: 0–59” />

<Element order = “7” name = “Position-information message type”
itemType = “protocolItem” dataType = “uint8_t”>
<ProtocolItem name = “Position Information Message Type Protocol”
type = “ordinaryProtocol” hasChecksum = “no”>
<FrameHeaderInfo count = “3” />
<ElementInfo count = “3”>
<Element order = “1” name = “Longitude” length = “4”
dataType = “uint32_t”
remarks = “real value = raw value × 0.000001” />
<Element order = “2” name = “Latitude” length = “4”
dataType = “uint32_t”
remarks = “real value = raw value × 0.000001” />
<Element order = “3” name = “Altitude” length = “2”
dataType = “uint16_t”
remarks = “real value = raw value × 0.1” />
</ElementInfo>
</ProtocolItem>
</Element>

<Element order = “8” name = “Type indicator” enumeration = “yes”>
<EnumDefinition count = “5”>
<Enum value = “0x00” description = “Type A” />
<Enum value = “0x01” description = “Type B” />
<Enum value = “0x02” description = “Type C” />
Enum value = “0x03” description = “Type D” />
<Enum value = “0x04” description = “Type E” />
</EnumDefinition>
</Element>

<Element order = “9” name = “Device ID” itemType = “stringItem”
length = “4” encoding = “UTF-8” />

<Element order = “10” name = “Target count N” itemType = “dynamicItem”
dynamicItemContent = “number of repeated target records”>
<Element order = “1” name = “Target flag” dynamic = “yes” />
<Element order = “2” name = “Target ID” dynamic = “yes” />
<Element order = “3” name = “Target status” dynamic = “yes” />
<Element order = “4” name = “Coordinate” dynamic = “yes”
processingFunction = “physicalValue” scaleFactor = “0.01” />
```

Listing 2. *Cont.*

```
<Element order = "5" name = "X-direction velocity" dynamic = "yes"
processingFunction = "physicalValue" scaleFactor = "0.1" />
</Element>
</ElementInfo>
<FrameTailInfo count = "0" />
</ProtocolItem>
</ProtocolInfo>
```

The example shows how the proposed method links semantic understanding and structural type identification to executable template construction. The nine-column table provides normalized field-level semantics, while the structural description result supplies higher-level organization information, including dynamic grouping, sub-protocol partitioning, enumeration mappings, and checksum parameters. These two intermediate outputs jointly enable the XML generator to construct a standardized protocol template that is consistent with the protocol item taxonomy.

5. Discussion

The experimental results show that the staged protocol construction pipeline based on large language models and type expert constraints has clear advantages in both resource interface protocol understanding and automatic template generation. At the protocol understanding level, the proposed method significantly improves main-table extraction accuracy and protocol item type recognition performance, especially on complex structures such as bit-field parsing, dynamic items, struct items, and sub-protocol items. At the XML level, the proposed method can not only generate XML files stably, but also significantly outperform rule-based and single-stage large language model methods in terms of node-structure consistency. These results indicate that, for highly structured technical documents with complex implicit semantics such as resource interface protocols, neither rule matching nor one-shot generation alone can simultaneously guarantee semantic correctness and engineering stability. Instead, the staged design of "semantic understanding–type recognition–structural integration–template generation" is more suitable for real engineering scenarios [20,21].

These findings are encouraging; however, the high metric values should be interpreted within the scope of the current 20-sample LVC protocol-table benchmark. This benchmark was designed to cover representative protocol item types and structural patterns observed in the collected LVC protocol documents, rather than to estimate performance over all possible protocol styles or industrial domains. In particular, the values of 1.0 for parser compatibility and XML generation success mainly indicate that the structured intermediate outputs can be accepted by the downstream program and converted into XML files without runtime failure.

In the current benchmark, the proposed method achieves a node consistency of 0.9478, indicating high but not perfect agreement with the gold-standard XML templates. Similarly, the overall type-recognition F1-score is 0.9769 rather than 1.0, although some individual protocol item categories achieve F1-scores of 1.0. Therefore, the high scores reported in this study should be regarded as evidence of effectiveness on the current LVC protocol-table benchmark, rather than as proof of universal generalization to all protocol styles or industrial domains.

From the perspective of scalability, the current benchmark verifies the feasibility of the proposed method under representative LVC protocol-table structures, but it does not fully reflect the difficulty of large-scale deployment across a much broader collection of

engineering protocol documents. When the number of protocol documents increases, the system may encounter more heterogeneous table headers, inconsistent field naming conventions, incomplete remarks, irregular auxiliary-table references, and more complicated combinations of dynamic items, struct items, branch items, and sub-protocol items. These factors may increase the difficulty of main-table identification, auxiliary-table association, protocol item type recognition, and XML structural alignment. Therefore, larger-scale application of the proposed method would require further expansion of the benchmark, more systematic cross-source evaluation, and stronger intermediate-result validation before the method can be claimed to be generally applicable to broader protocol collections.

At the same time, the experiments also reveal several challenging scenarios rather than complete failures of the overall pipeline. First, a few complex categories, such as branch elements, still have limited sample sizes in the current test set, which makes the corresponding category-level evaluation statistically unstable. Second, the node consistency of 0.9478 shows that the generated XML templates are highly consistent with the gold-standard templates, but small deviations still exist in fine-grained structural alignment. These deviations are mainly related to complex structural boundary reasoning, such as distinguishing repeated dynamic elements from fixed struct elements, determining the boundary of nested sub-protocols, and preserving the correct parent–child relationships in XML nodes. Third, auxiliary-table semantic mapping remains a challenging case when the meaning of a field depends on remarks, value-description tables, bit-definition tables, or other external explanatory tables. In these cases, the model must not only extract field-level attributes, but also correctly associate cross-table evidence with the corresponding protocol item. Fourth, the runtime anomalies observed in the single-stage method indicate that LLM outputs may become difficult to deploy when structured constraints are insufficient. Therefore, future applications should not rely only on semantic generation capability, but should also include stronger output-format constraints, intermediate-result validation, and XML-level consistency checking to improve engineering usability [22–25].

Another issue concerns the dependence on prompt engineering. The proposed method improves structural stability by decomposing protocol understanding into several expert-guided subtasks and by imposing explicit output-format constraints. However, this design also introduces additional prompt-design cost, API-call overhead, and prompt-version management requirements. The analysis of prompt sensitivity and cost shows that prompts for common field types are relatively reusable, whereas prompts for dynamic items, branch items, and nested sub-protocols are more sensitive to document style, table layout, and auxiliary-table organization. Therefore, although the prompt-engineering overhead is acceptable for engineering scenarios that prioritize structural correctness and XML-generation reliability, future work should further investigate automatic prompt validation, prompt drift detection, and lightweight prompt adaptation strategies for new protocol domains.

From an application perspective, the value of the proposed method lies not only in improving the automation level of protocol document processing, but also in providing a practical technical route for the rapid integration of heterogeneous LVC resources into a unified test system architecture [26]. By effectively linking protocol semantic understanding with standard template generation, the proposed method can significantly reduce the workload caused by manual item-by-item analysis and template filling, thereby supporting resource integration in complex simulation and testing environments. Future work can be further carried out in four directions: expanding the scale and diversity of the protocol benchmark test set to enhance coverage of extremely complex protocol types; conducting cross-source and cross-domain evaluations to further assess generalization capability; investigating finer-grained XML structural evaluation metrics and automatic repair mechanisms; and developing automatic prompt validation, prompt drift detection, and lightweight

prompt adaptation strategies to reduce prompt-engineering cost and improve deployment efficiency in practical engineering environments.

6. Conclusions

This paper addresses the problems that the analysis and template construction of heterogeneous LVC resource interface protocols still rely heavily on manual work, exhibit low processing efficiency, and are prone to errors. An intelligent construction method for resource interface protocol templates based on large language models is proposed. By constructing a staged pipeline of "protocol preprocessing–LLM-based semantic understanding–protocol item type identification experts–information integration and structured output–XML template generation", the proposed method realizes the automatic conversion from raw protocol text to standardized XML templates. Meanwhile, a protocol item taxonomy is established based on more than 30 protocol specification documents and over 5000 protocol items, thereby providing a knowledge basis for modeling complex protocol structures.

Experimental results show that the proposed LLM-Expert Pipeline outperforms both rule-based and single-stage large language model methods at the protocol understanding and structured extraction level as well as the end-to-end XML template generation level. On the current 20-sample LVC protocol-table benchmark, the full method achieves the best results in terms of main-table extraction accuracy, type-recognition F1-score, XML generation success rate, and node consistency, validating the effectiveness of the staged framework in complex protocol understanding, structural modeling, and engineering deployability. The ablation study further demonstrates that the semantic understanding module and the type expert module are the key factors contributing to the overall performance. Although the single-stage generation method possesses a certain degree of semantic capability, it shows clear deficiencies in structural stability and runtime reliability.

Overall, this work demonstrates the feasibility and application potential of large language models in the automatic construction of resource interface protocol templates and provides an effective solution for the rapid integration of heterogeneous resources into a unified test system. In future work, the scale and diversity of the benchmark will be further expanded, cross-source and cross-domain evaluations will be conducted, and more systematic analyses of prompt sensitivity, prompt transferability, prompt drift, and prompt-tuning cost will be performed. In addition, node-level evaluation and automatic repair mechanisms will be further investigated to improve the robustness and engineering practicality of the proposed method.

Author Contributions: Conceptualization, Y.Z. and Y.X.; methodology, Y.Z., D.W. and Y.X.; software, Y.Z. and D.W.; validation, Y.Z., D.W., B.X. and Y.H.; formal analysis, D.W.; data curation, D.W.; writing—original draft preparation, D.W.; writing—review and editing, Y.Z., D.W. and Y.X.; visualization, D.W.; supervision, C.W., G.D. and Y.X.; project administration, C.W., Y.X. and G.D. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Data Availability Statement: The original contributions presented in this study are included in the article. Further inquiries can be directed to the corresponding author.

Acknowledgments: The authors would like to thank all individuals who provided helpful discussions and technical support during the preparation of this study.

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

Abbreviations

The following abbreviations are used in this manuscript:

LVC	Live, Virtual, and Constructive
LLM	Large Language Model
XML	Extensible Markup Language
BCD	Binary-Coded Decimal
ASCII	American Standard Code for Information Interchange

References

1. *IEEE Std 1516-2010*; IEEE Standard for Modeling and Simulation (M&S) High Level Architecture (HLA)–Framework and Rules—Redline. IEEE: Piscataway, NJ, USA, 2010; pp. 1–38. [CrossRef]
2. 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. 4634–4642.
3. Shen, K.; Lu, J.; Yang, Y.; Chen, J.; Zhang, M.; Duan, H.; Zhang, J.; Zheng, X. HDiff: A Semi-Automatic Framework for Discovering Semantic Gap Attack in HTTP Implementations. In *Proceedings of the 2022 52nd Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*; IEEE: Piscataway, NJ, USA, 2022; pp. 1–13.
4. Song, J.; Cadar, C.; Pietzuch, P. SymbexNet: Testing Network Protocol Implementations with Symbolic Execution and Rule-Based Specifications. *IEEE Trans. Softw. Eng.* **2014**, *40*, 695–709. [CrossRef]
5. Moayyed, E.; Anumba, C.; Morteza, A. Systematic Analysis of Large Language Models for Automating Document-to-Smart Contract Transformation. *Autom. Constr.* **2025**, *175*, 106209. [CrossRef]
6. Liu, K.; Chakraborty, D.; Liggesmeyer, A.; Zeller, A. Synthesizing Precise Protocol Specs from Natural Language for Effective Test Generation. *arXiv* **2025**, arXiv:2511.17977. [CrossRef]
7. Deng, S.; Huang, R.; Zhang, M.; Cui, C.; Towey, D.; Wang, R. LRASGen: LLM-Based RESTful API Specification Generation. *ACM Trans. Softw. Eng. Methodol.* **2025**, *35*, 84. [CrossRef]
8. Phuong Nguyen, V.D.; Amir-Mohammadian, S. Large Language Models for Automated Network Protocol Testing: A Survey. In *Proceedings of the 2025 IEEE World AI IoT Congress (AIIoT)*; IEEE: Piscataway, NJ, USA, 2025; pp. 0814–0819.
9. 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*; Association for Computing Machinery: New York, NY, USA, 2022; pp. 4083–4091.
10. Jiang, S.; Evans-Yamamoto, D.; Bersenev, D.; Palaniappan, S.K.; Yachie-Kinoshita, A. ProtoCode: Leveraging Large Language Models for Automated Generation of Machine-Readable Protocols from Scientific Publications. *arXiv* **2023**, arXiv:2312.06241. [CrossRef] [PubMed]
11. Bhoite, H. AI-Driven Generation of Data Contracts in Modern Data Engineering Systems. *arXiv* **2025**, arXiv:2507.21056. [CrossRef]
12. Richter, C.; Wehrheim, H. Beyond Postconditions: Can Large Language Models Infer Formal Contracts for Automatic Software Verification? *arXiv* **2025**, arXiv:2510.12702. [CrossRef]
13. Wu, Z.; Chen, J.; Paton, N.W. Taxonomy Inference for Tabular Data Using Large Language Models. In *Proceedings of the European Semantic Web Conference*; Springer Nature: Cham, Switzerland, 2025.
14. Wang, D.; Raman, N.; Sibue, M.; Ma, Z.; Babkin, P.; Kaur, S.; Pei, Y.; Nourbakhsh, A.; Liu, X. DocLLM: A Layout-Aware Generative Language Model for Multimodal Document Understanding. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*; Ku, L.-W., Martins, A., Srikumar, V., Eds.; Association for Computational Linguistics: Bangkok, Thailand, 2024; pp. 8529–8548.
15. Appalaraju, S.; Jasani, B.; Kota, B.U.; Xie, Y.; Manmatha, R. DocFormer: End-to-End Transformer for Document Understanding. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*; IEEE: Piscataway, NJ, USA, 2021; pp. 993–1003.
16. Herzig, J.; Nowak, P.K.; Müller, T.; Piccinno, F.; Eisenschlos, J. TaPas: Weakly Supervised Table Parsing via Pre-Training. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*; Jurafsky, D., Chai, J., Schluter, N., Tetreault, J., Eds.; Association for Computational Linguistics: Stroudsburg, PA, USA, 2020; pp. 4320–4333.
17. Yao, S.; Liu, D.; Shen, Y.; Hu, J. CodeDocAgent: Leveraging Large Language Models for Accurate and Contextual Code Documentation. In *Proceedings of the 2025 IEEE International Symposium on Broadband Multimedia Systems and Broadcasting (BMSB)*; IEEE: Piscataway, NJ, USA, 2025; pp. 1–6.
18. Perot, V.; Kang, K.; Luisier, F.; Su, G.; Sun, X.; Boppana, R.S.; Wang, Z.; Wang, Z.; Mu, J.; Zhang, H.; et al. LMDX: Language Model-Based Document Information Extraction and Localization. In *Proceedings of the Findings of the Association for Computational Linguistics: ACL 2024*; Ku, L.-W., Martins, A., Srikumar, V., Eds.; Association for Computational Linguistics: Bangkok, Thailand, 2024; pp. 15140–15168.

19. Bhattacharyya, A.; Tripathi, A.; Das, U.; Karmakar, A.; Pathak, A.; Gupta, M. Information Extraction from Visually Rich Documents Using LLM-Based Organization of Documents into Independent Textual Segments. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*; Che, W., Nabende, J., Shutova, E., Pilehvar, M.T., Eds.; Association for Computational Linguistics: Vienna, Austria, 2025; pp. 17241–17256.
20. Fakhoury, S.; Kuppe, M.; Lahiri, S.K.; Ramananandro, T.; Swamy, N. 3DGen: AI-Assisted Generation of Provably Correct Binary Format Parsers. In *Proceedings of the 2025 IEEE/ACM 47th International Conference on Software Engineering (ICSE)*; IEEE: Piscataway, NJ, USA, 2025.
21. Wael, F.; Maklad, Y.; Hamdi, A.; Elsersy, W. An Agentic Flow for Finite State Machine Extraction Using Prompt Chaining. In *Proceedings of the 2025 Intelligent Methods, Systems, and Applications (IMSA)*; IEEE: Piscataway, NJ, USA, 2025; pp. 328–333.
22. Sheth, A.; Sheth, I.; Fritz, M. ProtocolLLM: RTL Benchmark for SystemVerilog Generation of Communication Protocols. *arXiv* **2025**, arXiv:2506.07945.
23. Duclos, M.; Fernandez, I.A.; Moore, K.; Mittal, S.; Zieglar, E. ModelForge: Using GenAI to Improve the Development of Security Protocols. In *International Symposium on Foundations and Practice of Security*; Springer Nature: Cham, Switzerland, 2025.
24. Mastouri, M.; Ksontini, E.; Kessentini, W. Making REST APIs Agent-Ready: From OpenAPI to MCP Servers for Tool-Augmented LLMs. *arXiv* **2025**, arXiv:2507.16044.
25. Zheng, M.; Xie, D.; Zhang, X. Large Language Models for Validating Network Protocol Parsers. In *2025 IEEE Security and Privacy Workshops (SPW)*; IEEE: Piscataway, NJ, USA, 2025.
26. Vieira Da Silva, L.M.; Köcher, A.; Gehlhoff, F. Beyond Formal Semantics for Capabilities and Skills: Model Context Protocol in Manufacturing. In *Proceedings of the 2025 IEEE 30th International Conference on Emerging Technologies and Factory Automation (ETFA)*; IEEE: Piscataway, NJ, USA, 2025; pp. 1–4.