2025 40th IEEE/ACM International Conference on Automated Software Engineering (ASE)

Bridging Natural Language and Formal Specification– Automated Translation of Software Requirements to LTL via Hierarchical Semantics Decomposition Using LLMs

Zhi Ma[1], Cheng Wen[2], Zhexin Su[1], Xiao Liang[1], Cong Tian[1*] , Shengchao Qin[1] and Mengfei Yang[3]
[1]School of Computer Science and Technology, Xidian University, Xi'an, China
[2]Guangzhou Institute of Technology of Xidian University, Guangzhou, China
[3]China Academy of Space Technology, Beijing, China
{mazhi, wencheng, qinshengchao}@xidian.edu.cn, {23031212487, xiaoliang}@stu.xidian.edu.cn,
ctian@mail.xidian.edu.cn, yangmf@bice.org.cn

***Abstract*—Automating the translation of natural language (NL) software requirements into formal specifications remains a critical challenge in scaling formal verification practices to industrial settings, particularly in safety-critical domains. Existing approaches, both rule-based and learning-based, face significant limitations. While large language models (LLMs) like GPT-4o demonstrate proficiency in semantic extraction, they still encounter difficulties in addressing the complexity, ambiguity, and logical depth of real-world industrial requirements. In this paper, we propose REQ2LTL, a modular framework that bridges NL and Linear Temporal Logic (LTL) through a hierarchical intermediate representation called *OnionL*. REQ2LTL leverages LLMs for semantic decomposition and combines them with deterministic rule-based synthesis to ensure both syntactic validity and semantic fidelity. Our comprehensive evaluation demonstrates that REQ2LTL achieves 88.4% semantic accuracy and 100% syntactic correctness on real-world aerospace requirements, significantly outperforming existing methods.**

***Index Terms*—formal specifications, large language models, formal verification, linear temporal logic, software requirement**

I. INTRODUCTION

The accurate and automated translation of natural language (NL) software requirements into formal language (FL) program specifications is crucial for leveraging formal verification techniques in industrial applications, particularly within safety-critical domains such as aerospace [1], operating systems [2], [3], compilers [4], and embedded controllers [5]. Linear Temporal Logic (LTL) is widely used to formally specify temporal behaviors of reactive and embedded systems due to its precision in expressing complex safety and liveness properties and the availability of powerful verification tools (*e.g.*, NuSMV [6], Spot [7]). However, the current industry practice predominantly relies on human experts with deep domain and formal reasoning knowledge to manually translate natural language requirements into LTL formulas. This process is both time-consuming and prone to errors [8]–[11].

Existing approaches aimed at automating NL-to-LTL translation are either rule-based or learning-based, each exhibiting significant drawbacks. Rule-based methods [12], [13] generally lack flexibility and are limited by variability and narrow scope. In contrast, learning-based approaches [14]–[16] require extensive labeled datasets and often struggle to generalize beyond their training examples. Recent advancements in large language models (LLMs), such as GPT-4o [17], have shown potential in related domains like code generation and logical inference [11], [18]–[22], yet directly applying these models to complex NL-to-LTL translation tasks remains problematic due to the implicit temporal semantics, deeply nested logical structures, and context-specific constraints inherent in industrial requirements.

Three primary challenges limit the effectiveness of applying LLMs directly to this translation task. First, natural language often conveys temporal semantics implicitly through nuanced expressions (*e.g.*, *will be set* or *unless*), which require deeper semantic interpretation beyond surface-level syntactic analysis. Second, industrial requirements often involve deeply nested logical constructs, including multiple conditionals, exceptions, and working mode-based constraints within single statements, posing significant challenges to the attention allocation and structural fidelity of LLMs. Third, LLMs are autoregressive and prone to error compounding: if a partial formula is generated incorrectly, subsequent completions are likely to deviate further from the intended semantics. These challenges are exacerbated by the absence of structural validation mechanisms in most LLM-based methods, resulting in silent failures that are difficult to detect or rectify.

To effectively address these issues, we propose a novel intermediate representation called *OnionL*, specifically designed to bridge the gap between natural language semantics and formal logic. *OnionL* is a tree-structured intermediate language that reflects the compositional syntax of LTL while maintaining semantic alignment with domain-specific expressions. It provides a hierarchical, structured representation of requirements by explicitly modeling *scopes*, *relations*, and *atomic propositions*. This structured abstraction serves as a semantic bridge, allowing LLMs to focus on local semantic extraction while delegating global logical composition to rule-based synthesis.

Building on the *OnionL* representation, we propose the REQ2LTL framework, which comprises two primary components: (1) the REQ2ONIONL module utilizes prompt-guided LLMs to decompose natural language requirements into struc-

*Corresponding author: Cong Tian.

2643-1572/25/$31.00 ©2025 IEEE
DOI 10.1109/ASE63991.2025.00104

TABLE I
Natural Language and Corresponding LTL Specifications

Natural Language	LTL Formula
Once red, the light cannot become green next.	$G(\text{red} \rightarrow X\neg\text{green})$
Once the light is red, it must remain red until it turns yellow.	$G(\text{red} \rightarrow \text{red}\, U\, \text{yellow})$
If b holds, next c holds until a holds or always c holds.	$G(\text{b} \rightarrow X((\text{c}\, U\, \text{a}) \vee G\text{c}))$
If a holds then c is true until b.	$G(\text{a} \rightarrow (\text{c}\, U\, \text{b}))$
Navigate to the green room while avoiding landmark 1.	$(F \text{ green}) \wedge G(\neg \text{ landmark } 1)$
Swing by landmark 1 before ending up in the red room.	$F(\text{landmark } 1 \wedge F \text{ red})$

tured *OnionL* trees hierarchically; (2) the OnionL2LTL module employs a deterministic, rule-based approach to validate and translate these *OnionL* structures into correct and standardized LTL formulas. The entire pipeline is fully automated, and it supports optional human-in-the-loop refinement via visualized intermediate structures.

Our evaluation shows that Req2LTL substantially surpasses existing state-of-the-art methods, such as NL2SPEC [23], NL2LTL [24], and NL2TL [16], on both academic benchmarks and real-world aerospace datasets. Specifically, Req2LTL achieves 88.4% semantic accuracy and 100% syntactic correctness, clearly demonstrating its efficacy in practical industrial contexts. Ablation studies confirm the importance of hierarchical decomposition and rule-based validation in handling complex, nested logic.

Our contributions are summarized as follows:

- **OnionL Intermediate Representation**: We introduce *OnionL*, a hierarchical intermediate language explicitly encoding temporal semantics, scopes, and logical relations. *OnionL* serves as a robust semantic bridge between natural language requirements and LTL formulas.
- **Hierarchical Semantic Decomposition**: We present a two-stage, prompt-driven decomposition algorithm (BuildingOnionL) leveraging LLMs. This systematic approach significantly improves semantic accuracy and structural integrity in the translation process.
- **Req2LTL Framework**: We present Req2LTL, an automated framework combining LLM-based semantic decomposition (Req2OnionL) and deterministic rule-based translation (OnionL2LTL), supported by automated validation. It achieves superior accuracy (88.4%) and perfect syntactic correctness (100%) compared to existing approaches, across both academic and industrial benchmarks.

II. Background and Motivation

A. Background

Linear Temporal Logic (LTL) is a formal modal logic extensively used to specify temporal properties over infinite execution paths. Formally, given a finite set of atomic propositions AP, LTL formulas follow the grammar:

$$\varphi ::= p \mid \neg\varphi \mid \varphi_1 \wedge \varphi_2 \mid \mathbf{X}\varphi \mid \mathbf{F}\varphi \mid \mathbf{G}\varphi \mid \varphi_1 \,\mathbf{U}\, \varphi_2$$

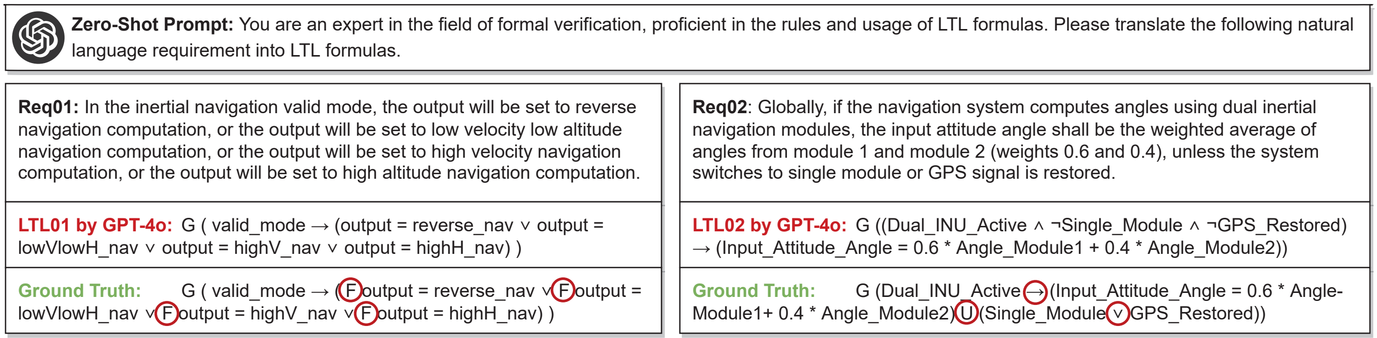
where $p \in AP$. The temporal operators in LTL convey critical aspects of system dynamics: $\mathbf{X}\varphi$ specifies that φ holds in the next state; $\mathbf{F}\varphi$ indicates φ eventually holds at some future state; $\mathbf{G}\varphi$ demands φ to always hold; and $\varphi_1\mathbf{U}\varphi_2$ requires φ_1 to hold φ_2 until becomes true.

Table I provides common examples from academic benchmarks [16], [23]–[26], where the natural language statements are typically concise, grammatically regular, and temporally explicit. The inclusion of keywords such as *always* or *if...then* allows for a direct correlation to standard LTL operators, facilitating a more regulated translation process. In contrast, requirements derived from safety-critical industrial systems exhibit a significantly higher level of complexity. They are generally longer, more deeply nested, and enriched with domain-specific terminology. Often, these requirements extend across multiple sentences, incorporating implicit assumptions and composite logical constructs.

B. Motivating Examples

Figure 1 presents two representative real-world cases where GPT-4o's translation yields significant discrepancies from the correct LTL specifications. The results in both cases are also consistent when employing NL2SPEC [23]. **Req01** demonstrates LLMs' struggle with implicit temporal constraints. The requirement states that in "*inertial navigation valid mode*," the system will eventually output one of four navigation computation types. While GPT-4o correctly maps the logical disjunction (∨), it misses the eventuality constraint (`F` operator). The generated LTL `G(valid_mode → output = ...)` incorrectly implies that the output must be set immediately upon entering the mode. In contrast, the ground truth `G(valid_mode → F output = ...)` accurately reflects that the system may take a finite time to determine the appropriate computation—a nuance critical in aerospace systems.

Similarly, in **Req02**, the requirement involves dual inertial navigation logic; the *unless* clause is misinterpreted as a simple negation rather than its intended meaning *until a condition occurs*. GPT-4o's output introduces three critical flaws: (1) Missing implicit temporal cues: it omits the until (`U`) operator, failing to specify that the weighted average holds only until termination conditions occur. (2) Flattened logical structure that loses nesting and scope boundaries: the ground truth `G(Dual_INU_Active → ... U (Single_Module ∨ GPS_Restored)` correctly prioritizes `Dual_INU_Active` as the sole trigger. While the GPT-4o erroneously treats all conditions as co-occurring prerequisites, due to the flat logical structure fails to capture the hierarchical relationship: `G(...→...U(...∨...)`. This error stems from the inherent tendency



Zero-Shot Prompt: You are an expert in the field of formal verification, proficient in the rules and usage of LTL formulas. Please translate the following natural language requirement into LTL formulas.

Req01: In the inertial navigation valid mode, the output will be set to reverse navigation computation, or the output will be set to low velocity low altitude navigation computation, or the output will be set to high velocity navigation computation, or the output will be set to high altitude navigation computation.

LTL01 by GPT-4o: G (valid_mode → (output = reverse_nav ∨ output = lowVlowH_nav ∨ output = highV_nav ∨ output = highH_nav))

Ground Truth: G (valid_mode → (F output = reverse_nav ∨ F output = lowVlowH_nav ∨ F output = highV_nav ∨ F output = highH_nav))

Req02: Globally, if the navigation system computes angles using dual inertial navigation modules, the input attitude angle shall be the weighted average of angles from module 1 and module 2 (weights 0.6 and 0.4), unless the system switches to single module or GPS signal is restored.

LTL02 by GPT-4o: G ((Dual_INU_Active ∧ ¬Single_Module ∧ ¬GPS_Restored) → (Input_Attitude_Angle = 0.6 * Angle_Module1 + 0.4 * Angle_Module2))

Ground Truth: G (Dual_INU_Active → (Input_Attitude_Angle = 0.6 * Angle-Module1+ 0.4 * Angle_Module2) U (Single_Module ∨ GPS_Restored))

Fig. 1. **Examples showcasing the discrepancy between LLM generated LTL and ground truth for complex requirements.** The **red-circle** indicates errors in the LLM's output due to missing details, contrasted with the manually corrected specification.

of LLMs to prioritize syntactic translation over profound logical reasoning. (3) Accumulation of errors and lack of validation: the formula produced by GPT-4o initially included `Dual_INU_Active` ∧ ¬`Single_Module` ∧ ¬`GPS_Restored`, which omitted the implies relationship, and subsequent generations were based on this deviation, resulting in further semantic deviation. This mismatch leads to "silent failures", where the formula is syntactically valid but semantically incorrect, as highlighted by the red circles in Figure 1.

To further investigate the limitations of using LLMs for translating natural language into LTL, we collected 112 natural language requirements from two real-world aerospace systems and conducted detailed empirical experiments. Under zero-shot prompting, GPT-4o generates only 49 semantically correct LTL formulas, highlighting a broader difficulty in capturing embedded temporal semantics when cues are implicit or structurally distant from their target propositions. However, the model demonstrates proficiency in extracting atomic propositions, achieving a recall rate of 98.5%. These observations underscore a significant limitation of current LLM-based translation methods: while LLMs excel at extracting atomic propositions, they struggle with global logical synthesis. In particular, LLMs are adept at recognizing isolated facts but lack the structured reasoning needed to correctly compose and validate nested temporal and logical relationships, especially when implicit cues are present.

C. Towards a Two-stage, Structured Translation Approach

To effectively overcome the above inherent limitations, a structured, hierarchical approach is crucial, combining the strengths of LLMs in local semantic interpretation and atomic proposition extraction with deterministic rule-based mechanisms for global logical synthesis.

Motivated by these insights, we first introduce *OnionL*, a hierarchical intermediate representation explicitly designed to bridge the semantic gap between natural language and LTL. By explicitly encoding temporal semantics and logical relations in a structured intermediate representation, we can effectively reduce the cognitive load on LLMs, allowing them to focus on accurate semantic parsing rather than complex logical composition. Leveraging *OnionL* representation, our proposed REQ2LTL framework (Section III) systematically decomposes requirements via the REQ2ONIONL module (Section IV). It deterministically translates the resulting structured representations into accurate LTL formulas using the ONIONL2LTL module (Section V), thereby ensuring semantic accuracy and structural integrity.

III. GLOBAL VIEW OF THE REQ2LTL FRAMEWORK

Figure 2 presents an overview of the REQ2LTL framework, which enables the automatic translation of complex NL requirements into formal LTL specifications through a structured intermediate representation called *OnionL*. Through staged modeling and a rule-based constraint mechanism, the framework ensures semantic consistency and structural correctness. The framework consists of two functionally independent yet complementary modules: REQ2ONIONL and ONIONL2LTL.

A. Constructing the Structured "Onion" – REQ2ONIONL

The REQ2ONIONL (Section IV) module is responsible for transforming free-form NL requirements into structured *OnionL* expressions. This module employs an LLM as its generative engine, with the generation process constrained and guided by a knowledge repository and chain of thought.

Knowledge Repository (Section IV-A). Within this repository, the syntactic structure and semantic composition of *OnionL* are formally defined as a recursive linguistic system consisting of three types of semantic units: scopes, relations, and atomic propositions. The overall structure is mapped onto a tree-based syntactic framework.

Chain-of-Thought (Section IV-B). Our REQ2ONIONL integrates a Chain-of-Thought prompting algorithm named BUILDING-ONIONL. This algorithm operates in two stages. The first stage performs macro-structural extraction, identifying global temporal or pattern-based scopes. The second stage conducts recursive decomposition and atomic proposition normalization, ultimately generating a logically complete and semantically faithful *OnionL* JSON.

B. Decomposing the Structured "Onion" – ONIONL2LTL

The ONIONL2LTL (Section V) module takes the *OnionL* JSON and translates it into a well-formed LTL formula through a combination of validation and rule-based synthesis.

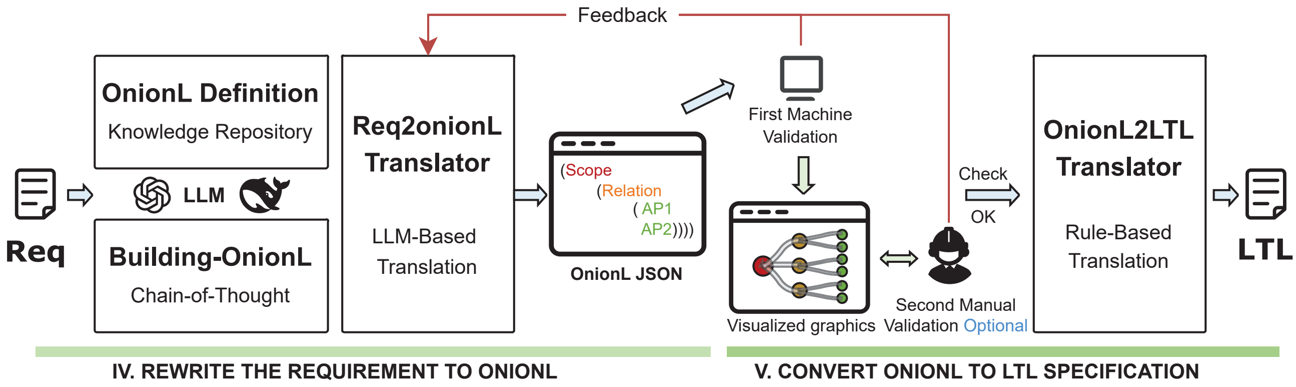


Fig. 2. **Overview of our proposed Req2LTL framework pipeline.** Given a natural language requirement, the system incrementally constructs a compositional *OnionL* tree via semantic prompting and structural parsing. The intermediate representation undergoes validation (machine and optional human-in-the-loop) and is then translated into an LTL formula by a rule-based engine.

Dual Validation (Section V-A). First, machine validation performs a depth-first traversal of the *OnionL* tree, validating scope-clause pairing rules node by node. It checks the arity and type validity of unary and binary operators and identifies redundant chains, undefined operations, or logical conflicts. Second, for manual validation, considering potential sentence ambiguity and incomplete requirements in safety-critical scenarios, OnionL2LTL supports automatically rendering OnionL JSON into Mermaid [27] formatted visual tree diagrams. This facilitates engineers in conducting semantic inspections efficiently, promoting a "human-in-the-loop" feedback and validation cycle.

Rule-Based Translation (Section V-B). Upon completing the validation, the OnionL2LTL module starts a rule-based translation into LTL. Each scope node is mapped to the corresponding unary operator, each relation node is converted into a binary operator, and each atomic proposition is reconstructed into a predicate expression according to its semantic label. This process is deterministic and structure-preserving, ensuring that each validated *OnionL* JSON corresponds to a distinct LTL formula that is both syntactically and semantically correct.

Together, Req2onionL and OnionL2LTL form a pipeline that not only achieves high translation accuracy but also introduces transparency, modularity, and verifiability into the process of deriving LTL specifications from NL.

IV. Rewrite the Requirement to OnionL

A. Design of the OnionL Intermediate Language

> *"Onions have layers. You get it? We both have layers."*
> *— Shrek (2001)*

This humorous line captures the core idea behind *OnionL*: natural language requirements, like onions, have layers. These layers represent nested semantic structures—temporal scopes, logical relations, and predicate-level facts—that must be preserved in formal specification.

Translating such requirements into LTL is inherently a structure-preserving task. While LTL uses a recursive grammar over logical and temporal operators, industrial requirements consistently follow **semantically layered patterns** due to strict demands on determinism, correctness, and timing.

To systematically capture these patterns, we introduce *OnionL*, a hierarchical intermediate language that decomposes requirements into three compositional elements: *scopes*, *relations*, and *atomic propositions*. Each requirement is represented as a tree built over these constructs, enabling deterministic parsing and rule-based translation into LTL. *OnionL* thus serves as a semantic bridge between informal natural language and formal temporal logic.

Atomic Propositions (APs) represent predicate-level facts about system behavior, such as sensor states, threshold conditions, or numeric assignments. Each AP is annotated with four semantic subfields to support symbolic interpretation:

- `Com`: component or subsystem identifier;
- `Var`: variable name or symbolic constant;
- `Rel`: relational operator (*e.g.*, $=$, $>$, $\leq$);
- `Formula`: numeric value or arithmetic expression.

These subfields are combined into canonical patterns, such as comparisons (`Var Rel Formula`) and assignments (`Var` $=$ `Formula`).

Scopes define the contextual boundary in which a requirement holds. They correspond to unary operators in LTL and can be:

- *Temporal scopes*, *e.g.*, `Globally`, `Eventually`, or `Next`;
- *Mode scopes*, which capture operational modes (*e.g.*, *"in navigation valid mode"*) and are always interpreted as antecedents in implications.

Relations define logical or temporal dependencies between subformulas. OnionL supports:

- *Logical relations*: conjunction ($\land$), disjunction ($\lor$), implication ($\rightarrow$);
- *Temporal relations*: variants of the `Until` operator.
  - *Basic precedence* – one condition eventually follows another;
  - *Sustained precedence* – one condition must hold continuously until another occurs.

Recursive Grammar. Let AP be the set of atomic propositions, SC the set of unary scope operators, and RE the set of binary relation operators. We denote $\varphi_{AP} \in AP$, $\varphi_{SC} \in SC$, and $\varphi_{RE} \in RE$ accordingly. An *OnionL* expression φ is defined recursively as:

(Base case)	$\varphi ::= \varphi_{AP} \in AP$
(Scope application)	$\varphi ::= \varphi_{SC}(\varphi_1)$
(Relational composition)	$\varphi ::= \varphi_1 \, \varphi_{RE} \, \varphi_2$
(Nested combination)	$\varphi ::= \varphi_{SC}(\varphi_1) \, \varphi_{RE} \, \varphi_{SC}(\varphi_2)$

To avoid ambiguity, all relation operators are left-associative, and scopes bind more tightly than relations. For instance, the *OnionL* tree for $G(F(p) \vee q)$ would be placed `Globally` at the root and `Eventually` applied to p as a nested child of the left operand.

Illustrative Example. Consider the requirement: "*In valid mode, if the temperature exceeds 50, eventually the warning light is turned on.*" Its corresponding *OnionL* expression is:

```
(Globally
    (Implies
        (Atomic: "workmode = valid")
        (Eventually
            (Implies
                (Atomic: "temperature > 50")
                (Atomic: "warning = ON")))))
```

This structure reflects the following semantic layering. A top-level `Globally` scope encapsulates the entire implication; The antecedent is a condition based on the system working mode; The consequent contains a nested implication under the `Eventually` operator. Such structured representation ensures that both temporal and conditional semantics are preserved and faithfully encoded in a compositional form. By compositionality we mean that LTL has an inductively defined syntax where complex formulas are built from subformulas via unary or binary operators, and its semantics is defined per operator. Hence, the meaning of a formula is determined compositionally from the meanings of its subformulas.

B. Two-Stage Decomposition Algorithm

We propose Building-OnionL, a hierarchical prompt-driven algorithm that transforms natural language requirements into structured *OnionL* trees. The algorithm operates in two semantic stages. **Stage I**: Macro-Structure Extraction. Identify the global semantic scope (Temporal or Mode) and construct the top-level *OnionL* node. **Stage II**: Recursive Clause Decomposition. Decompose the remaining clause into nested scopes, relations, and atomic propositions until a fully structured tree is obtained. This staged strategy mirrors the compositional form of LTL and enforces semantic anchoring prior to structural expansion, enabling faithful alignment between requirement semantics and formal representations. Algorithm 1 details its formal pseudo code, which comprises two stages and six steps.

Step 1: Scope Identification and Clause Separation. The model first identifies the overarching semantic scope of the requirement. This includes: Temporal scopes and Mode scopes. If no explicit marker is present, a default `Globally` scope is assumed. The remaining part of the sentence is separated as the main clause to be recursively processed.

Step 2: Scope Type Analysis and Top-Level Construction. If a temporal scope is detected, it is assigned as the root of the *OnionL* tree, with the remaining clause as its child. If the clause is atomic, the algorithm skips recursive decomposition and proceeds to Step 6. For mode scopes, the scope is interpreted as a logical antecedent, and the root is set to `Globally`, forming an implication from the mode condition to the consequent behavior. If no scope is found, the main clause is wrapped by `Globally` by default.

Step 3: Unary Operator Identification and Extraction. The model examines whether the current clause contains a unary temporal operator. If found, this operator becomes a new parent node, and the remaining clause is recursively parsed as its child node.

Step 4: Binary Logical Operator Decomposition. If the clause contains a binary logical connective, the clause is split into two subtrees, with the operator as the parent node. Each part is recursively processed.

Step 5: Atomicity Judgment and Recursive Termination. If neither unary nor binary operators are detected, the model evaluates whether the clause is atomic. If it is, the clause is encapsulated as a leaf node. Otherwise, the model devises a refined decomposition plan and reapplies Steps 3 through 5 to the revised clause.

Step 6: Semantic Reduction and AP Normalization. Once a clause is confirmed to be atomic, the final step performs semantic normalization. The clause is rewritten into one of the five predefined atomic proposition patterns, explicitly populating the `Com`, `Var`, `Rel`, and `Formula` fields. This ensures machine-checkable alignment with the engineering semantics expected by downstream verification tools.

C. The Req2OnionL Translator

The Req2OnionL translator forms the front-end of the Req2LTL framework. It transforms natural language requirements into structured *OnionL* trees by prompting LLMs under formal guidance. We divide this process into two key components. We treat the *OnionL* representation introduced in Section IV-A as a formal knowledge base. It defines the compositional syntax and semantic elements that serve as structural priors to constrain the generation space and ensure syntactic correctness. We utilize the Building-OnionL algorithm, introduced in Section IV-B, as a chain-of-thought prompting strategy. It guides the model to perform structured reasoning in two stages: first identifying the global scope, then recursively decomposing the sentence into a semantically faithful tree. To further improve robustness, we introduce grammar-constrained prompt templates along with a few task-specific few-shot examples. These examples capture common patterns in industrial requirements—such as mode-triggered behavior, temporal constraints, and threshold logic—and help the model generalize across structural variations. The output is a machine-readable *OnionL* JSON object that preserves the hierarchical semantics of the requirement. This serves as input

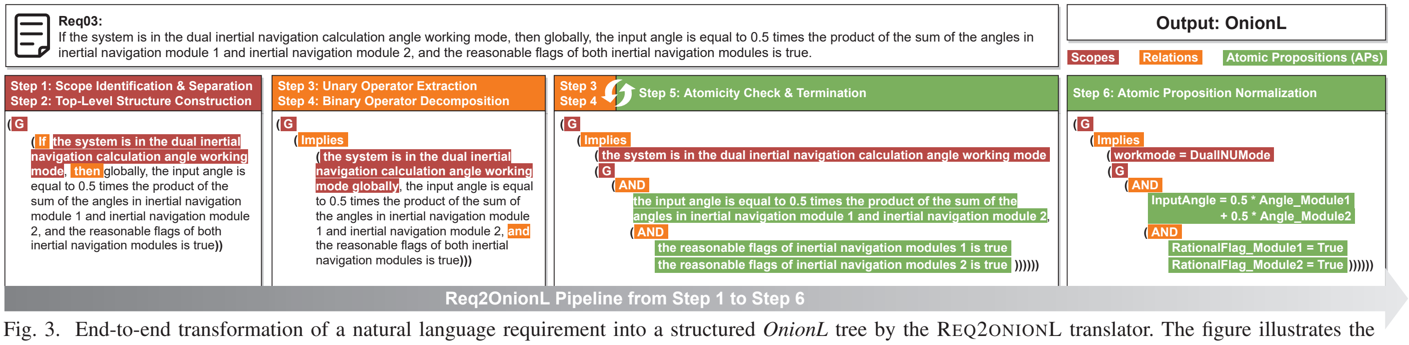

Req03:
If the system is in the dual inertial navigation calculation angle working mode, then globally, the input angle is equal to 0.5 times the product of the sum of the angles in inertial navigation module 1 and inertial navigation module 2, and the reasonable flags of both inertial navigation modules is true.
Output: OnionL
Scopes
Relations
Atomic Propositions (APs)
Step 1: Scope Identification & Separation
Step 2: Top-Level Structure Construction
(G
(If the system is in the dual inertial navigation calculation angle working mode, then globally, the input angle is equal to 0.5 times the product of the sum of the angles in inertial navigation module 1 and inertial navigation module 2, and the reasonable flags of both inertial navigation modules is true))
Step 3: Unary Operator Extraction
Step 4: Binary Operator Decomposition
(G
(Implies
(the system is in the dual inertial navigation calculation angle working mode globally, the input angle is equal to 0.5 times the product of the sum of the angles in inertial navigation module 1 and inertial navigation module 2, and the reasonable flags of both inertial navigation modules is true)))
Step 3
Step 4
Step 5: Atomicity Check & Termination
(G
(Implies
(the system is in the dual inertial navigation calculation angle working mode
(G
(AND
the input angle is equal to 0.5 times the product of the sum of the angles in inertial navigation module 1 and inertial navigation module 2,
(AND
the reasonable flags of inertial navigation modules 1 is true
the reasonable flags of inertial navigation modules 2 is true))))))
Step 6: Atomic Proposition Normalization
(G
(Implies
(workmode = DualINUMode
(G
(AND
InputAngle = 0.5 * Angle_Module1
+ 0.5 * Angle_Module2
(AND
RationalFlag_Module1 = True
RationalFlag_Module2 = True))))))
Req2OnionL Pipeline from Step 1 to Step 6

Fig. 3. End-to-end transformation of a natural language requirement into a structured *OnionL* tree by the REQ2ONIONL translator. The figure illustrates the full execution trace of the BUILDING-ONIONL algorithm, from top-level scope extraction to atomic proposition normalization. The resulting *OnionL* structure captures the hierarchical semantics of the input and enables deterministic translation into formal LTL.

Algorithm 1: BUILDING-ONIONL

```
Input: Natural language requirement $\mathcal{R}$
Output: Structured OnionL expression $\varphi$
// Stage I: Macro-Structure Extraction
  // Step 1: Scope Identification and Clause Separation
  scope, clause $\leftarrow$ EXTRACTSCOPE($\mathcal{R}$);
if scope is Temporal then
    // Step 2: Temporal scope $\rightarrow$ root node
    if ISATOMICCLAUSE(clause) then
        // Step 6: Atomic clause, normalize and terminate
        return NORMALIZEAP(clause)
    $\varphi \leftarrow$ scope(DECOMPOSECLAUSE(clause));
else
    if scope is Mode-based then
        // Step 2: Mode scope $\rightarrow$ G(antecedent $\rightarrow$ consequent)
        $\varphi \leftarrow$ G(scope $\rightarrow$ DECOMPOSECLAUSE(clause));
    else
        // Step 2 (fallback): Wrap clause with G
        $\varphi \leftarrow$ G(DECOMPOSECLAUSE(clause));
// Stage II: Recursive Clause Decomposition and AP Normalization
  Function DecomposeClause($c$):
    if HASUNARYSCOPE($c$) then
        // Step 3: Unary scope $\rightarrow$ parent
        scope, $c'$ $\leftarrow$ EXTRACTUNARYSCOPE($c$);
        return scope(DecomposeClause($c'$))
    if HASBINARYRELATION($c$) then
        // Step 4: Binary relation $\rightarrow$ subtrees
        $c_1, c_2$, rel $\leftarrow$ EXTRACTBINARYRELATION($c$);
        return DecomposeClause($c_1$) rel DecomposeClause($c_2$)
    if ISATOMICCLAUSE($c$) then
        return NORMALIZEAP($c$)// Step 5: Atomic clause, stop
    refined $\leftarrow$ REFINEIMPLICITSTRUCTURE($c$);
    return DecomposeClause (refined)// Step 5 (fallback): Refine
```

to downstream validation and formal translation. Figure 3 shows an example of the end-to-end result. As illustrated, the construction mirrors Algorithm 1 step by step: (1) detecting the global scope, (2) building the root and separating the main clause, (3) extracting unary operators, (4) splitting clauses by binary relations, (5) performing atomicity checks, and (6) normalizing atomic propositions. The right panel shows the resulting *OnionL* JSON aligned with the original requirement.

V. CONVERT ONIONL TO LTL SPECIFICATION

A. Validation of the OnionL

First Machine Validation. The system performs a recursive, depth-first traversal of the *OnionL* to verify structural well-formedness. Each node is checked against its expected role in LTL: unary operators must have exactly one valid child, binary operators must have two, and atomic propositions must occur only at the leaves. In addition, the system checks for *OnionL*-specific composition constraints. Atomic propositions are expected to contain one or more well-formed subfields—such as `Com`, `Var`, `Rel`, and `Formula`—depending on the semantics of the requirement. The validator checks for compatibility among these subfields and flags missing or contradictory combinations where applicable; scope nesting is validated against a finite set of legal patterns. The validator detects redundant or malformed constructs, such as deeply nested `AND` chains, and rewrites them into canonical left-associative forms. Violations are reported with detailed diagnostic logs, including error type, tree path, and suggested fixes.

Second Manual Validation. For safety-critical requirements, human-in-the-loop validation is strongly recommended. The *OnionL* structure is visualized as a directed graph using Mermaid [27] syntax, as shown in Figure 4. This graphical representation exposes the hierarchical semantics inferred by the model in a readable and structured format, allowing engineers to inspect scopes, logical relationships, and atomic conditions efficiently. When semantic inconsistencies or structural errors are identified, engineers can directly modify the *OnionL* tree or provide feedback to the system, prompting the REQ2ONIONL module to regenerate a corrected structure. Once verified, the validated *OnionL* is passed to the rule-based translator for deterministic LTL synthesis.

B. Rule-Based Translation to LTL

Once the *OnionL* structure passes validation, it is forwarded to the ONIONL2LTL translator—a dedicated module responsible for converting structured trees into standard LTL formulas. This module implements a set of fixed, rule-based mappings that deterministically translate each semantic element of the tree into its corresponding LTL syntax. Formally, we define a total function T : *OnionL* $\rightarrow$ *LTL* that maps each node in the tree to its corresponding logical form.

The translation proceeds recursively in a post-order traversal of the *OnionL* tree. For each node, the algorithm first processes all of its children before applying the corresponding transformation at the parent level. For unary nodes such as `Globally`,

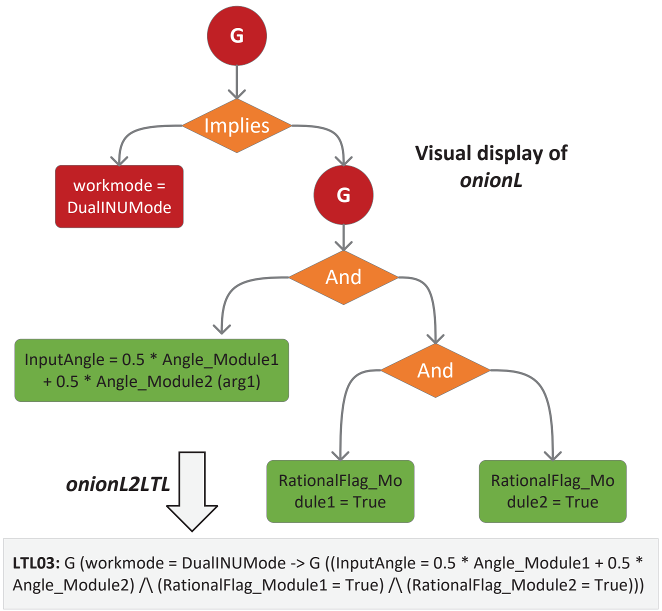

G
Implies
workmode = DualINUMode
G
Visual display of onionL
And
InputAngle = 0.5 * Angle_Module1 + 0.5 * Angle_Module2 (arg1)
And
onionL2LTL
RationalFlag_Module1 = True
RationalFlag_Module2 = True
LTL03: G (workmode = DualINUMode -> G ((InputAngle = 0.5 * Angle_Module1 + 0.5 * Angle_Module2) /\ (RationalFlag_Module1 = True) /\ (RationalFlag_Module2 = True)))

Fig. 4. Visualized *OnionL* structure for human-in-the-loop validation. The diagram presents the model's inferred semantics in a structured form, making it easier for engineers to identify semantic errors. Once validated, it is deterministically translated into an LTL formula via rule-based synthesis.

`Eventually`, or `Next`, the child node is fully translated first, after which the appropriate temporal operator is applied. For binary nodes such as `AND`, `OR`, or `Implies`, both child nodes are recursively translated before being combined using logical connectives such as $\wedge$, $\vee$, or $\rightarrow$. For atomic propositions at the leaves, the expression is directly reconstructed from the subfields `Com`, `Var`, `Rel`, and `Formula`, yielding a flattened predicate representation.

All translation rules in the ONIONL2LTL module are deterministic and structure-preserving: each well-formed *OnionL* tree corresponds to exactly one valid LTL formula. Provided that the *OnionL* representation faithfully captures the semantics of the original requirement, the resulting LTL is guaranteed to be both syntactically correct and semantically consistent. This rule-driven translation approach enables seamless integration with existing model checkers such as NuSMV [28], while Spot [7] is used as an LTL/ω-automata manipulation and parsing framework. It also supports automated batch conversion of requirements into formal specifications, making the framework scalable and robust for industrial verification workflows.

VI. EVALUATION

To assess the effectiveness of the proposed REQ2LTL framework, we conduct a comprehensive evaluation aimed at answering the following research questions:

RQ1. How does REQ2LTL perform on standard academic benchmarks compared to existing state-of-the-art NL-to-LTL translation methods?

RQ2. To what extent can REQ2LTL accurately and reliably translate real-world industrial requirements into semantically correct and syntactically valid LTL specifications?

RQ3. What is the contribution of the *OnionL* structure, the BUILDING-ONIONL algorithm, and the validation feedback mechanism to the overall performance?

RQ4. Are structural challenges in industrial requirements (*e.g.*, temporal ambiguity, nested logic) systematic and significant? How effectively does REQ2LTL address and mitigate these complexities?

A. Experimental Setup

1) Datasets: We evaluate two categories of data.

Academic Benchmark: To assess cross-domain generalization, we evaluate REQ2LTL on a public benchmark comprising three curated subsets: *Circuit*, *Navigation*, and *Office Email*, originally proposed in prior work [16], [24]–[26]. These requirements are shorter, less ambiguous, and syntactically regular, typically mapping directly to standard LTL constructs. The benchmark adopts a *lifted* representation in which all atomic propositions are abstracted into placeholders (*e.g.*, `Prop1`, `Prop2`). REQ2LTL accommodates this abstraction format by applying placeholder substitution during its ONIONL2LTL post-processing stage.

Industrial Dataset: Our dataset is built from requirement documents provided by our industrial aerospace partners. These requirements originate from two critical spacecraft systems: a *sun-search controller* system, which adjusts solar panels or spacecraft attitude to track the Sun, and a *propulsion management* system, which governs thruster activation and fuel flow during orbital maneuvers and attitude adjustments. In total, we collect 112 requirements covering diverse operational scenarios, including initialization, attitude determination, anomaly handling, and fault tolerance. Compared to academic datasets, these industrial specifications are characterized by longer sentence lengths (avg. 43.7 tokens), deeper logical nesting (over 60% with $\geq$ 2 layers), and domain-specific vocabulary grounded in hardware and control logic. A detailed structural complexity analysis is presented in Section VI-E. To ensure high-quality annotations, each requirement was independently translated into an LTL formula by two annotators. These annotators were selected from a team of five experts, including two aerospace engineers and three formal verification researchers. Discrepancies between the two annotations were resolved through expert discussion, and all finalized formulas were reviewed by a senior engineer to ensure semantic fidelity and consistency. During this process, we did encounter certain requirements that either involved explicit quantitative timing (e.g., response within 5 seconds) or lacked any temporal constraint. Since such requirements fall outside the expressive power of LTL, they were excluded from our benchmark to maintain semantic compatibility. The resulting set of 112 validated Req–LTL pairs constitutes our gold dataset and serves as the reference standard for evaluation throughout this study.

2) Baselines: We compare REQ2LTL with several baseline methods across two dimensions: model backend and generation strategy. For model backends, we choose two recent and representative large language models: GPT-4o [17] and

TABLE II
PERFORMANCE COMPARISON ON ACADEMIC BENCHMARKS (LIFTED)

Method	Domain	Binary Acc. (%)	BLEU Score
Req2LTL	Circuit	95.30	**0.97**
	Navigation	94.50	**0.97**
	Office Email	**96.70**	**0.98**
NL2TL	Circuit	**96.20**	0.96
	Navigation	**96.50**	**0.97**
	Office Email	96.00	0.96
NL2LTL	Circuit	90.50	0.93
	Navigation	89.90	0.93
	Office Email	90.80	0.93
NL2SPEC	Circuit	90.80	0.93
	Navigation	89.70	0.93
	Office Email	91.20	0.93

DeepSeek-V3 [29]. For generation strategies, we consider four methods with increasing levels of structural guidance. *Zero-Shot Prompt* represents a direct instruction method that asks the model to translate a requirement into LTL without templates or intermediate representations. NL2LTL [24] is a Python package developed by IBM Research that uses LLMs to translate NL instructions into LTL formulas. NL2SPEC [23] adopts a template-guided prompting approach that builds LTL expressions in stages, emphasizing interpretability and semantic traceability. NL2TL [16], originally designed for lifted STL translation, is adapted here as a prompt-based baseline for abstracted NL-to-LTL generation. We retain its prompt structure and abstraction convention, applying position-consistent placeholders to both inputs and references.

3) Evaluation Metrics: To comprehensively evaluate performance, we employ a multi-metric framework that integrates automated tools with expert review, based on: (1) *LTL Syntax Validity:* assesses whether the generated formula adheres to formal syntax. We use Spot to parse each formula; a formula is valid if it can be compiled into a Büchi automaton. (2) *Exact Match Accuracy:* measures semantic and logical equivalence to the reference. Two experts independently assess each pair, and a match is accepted only upon agreement. (3) *Atomic Proposition Recall:* evaluates coverage of key atomic propositions by comparing the AP sets of the output and the gold standard. (4) *BLEU Score:* estimates token-level structural similarity. To reduce lexical bias, atomic propositions are abstracted before scoring, focusing the metric on syntactic alignment. All methods are tested on the same dataset under consistent procedures to ensure fairness. Importantly, all results reported in RQ1 and RQ2 were obtained from a fully automated pipeline, with the optional manual validation feature disabled. This ensures that the reported performance metrics reflect the framework's automated capability without any human intervention.

B. RQ1. Performance on Academic Benchmarks

We evaluate the generalization performance of REQ2LTL on three widely used academic benchmarks—*Circuit*, *Navigation*, and *Office Email*—using GPT-4o as the backend. Two automatic metrics are used: *Binary Accuracy* measures exact string-level matches between generated and reference formulas without normalization or semantic equivalence checking, ensuring a strict evaluation of structural correctness. *BLEU Score*, in contrast, offers a softer measure of syntactic similarity by evaluating token-level overlap, emphasizing partial alignment and structural fluency. For academic benchmarks, we report only binary accuracy and *BLEU* because the formulas are short, syntactically regular, and use abstract placeholders, making *AP Recall* less informative. Moreover, binary accuracy already reflects both syntactic and emantic correctness in these settings.

As shown in Table II, REQ2LTL achieves strong and consistent performance across all three domains. It attains Binary Accuracy scores of 95.3%, 94.5%, and 96.7% respectively, along with BLEU scores ≥ 0.97—higher than all other methods. These results confirm that our hierarchical, two-stage translation framework generalizes effectively even in less ambiguous settings. Notably, its performance remains steady despite variations in structure and content across datasets. Furthermore, our comparison of REQ2LTL with three representative baselines demonstrates superior performance in BLEU scores comprehensively. NL2TL achieves comparable binary accuracy (96.0–96.5%) but exhibits slightly lower BLEU scores, suggesting minor structural drift. NL2SPEC and NL2LTL perform similarly in both metrics, with binary accuracy around 90% and BLEU scores near 0.93, reflecting moderate semantic precision but weaker syntactic control compared to REQ2LTL and NL2TL.

The answer to RQ1: REQ2LTL These results indicate that REQ2LTL not only retains high precision in simplified formalization tasks but also outperforms existing prompting baselines in syntactic fidelity. Compared to other existing methods, its modular architecture and intermediate representation (*OnionL*) allow robust alignment between natural language and logic, even in abstracted settings. On academic benchmarks, REQ2LTL performs comparably to NL2TL; our primary gains arise on semantically complex industrial requirements.

C. RQ2: Performance on Industrial Requirements

We evaluate the effectiveness of REQ2LTL on a real-world industrial dataset comprising 112 requirements from aerospace control systems. This dataset features domain-specific terminology, complex conditional logic, and implicit temporal semantics, posing significantly greater challenges than academic benchmarks.

As shown in Table III, REQ2LTL substantially outperforms all baseline prompting methods across both models. Under GPT-4o, it achieves an exact match rate of **88.4%**, full **100.0%** syntax validity, and a BLEU score of **0.96**, far surpassing the closest alternative (NL2TL at 65.2% exact match and 0.85 BLEU). Similarly, when paired with DeepSeek-V3, REQ2LTL maintains strong performance with 86.6% exact match and identical BLEU (0.96), indicating consistent robustness across LLM backends.

Zero-shot prompting yields poor exact match rates (43.8% with GPT-4o and 34.8% with DeepSeek-V3), highlighting the inadequacy of unguided LLMs in handling complex industrial

TABLE III
COMPARISON OF PROMPTING STRATEGIES AND ABLATION STUDY ON REQ2LTL FRAMEWORK. **BOLD** AND UNDERLINED VALUES INDICATE THE BEST AND SECOND-BEST RESULTS, RESPECTIVELY.

Setting	Exact Match (%)	Syntax Validity (%)	AP Recall (%)	BLEU Score
Prompt Strategy Comparison				
GPT-4o + Zero-Shot	43.8	89.3	98.5	0.76
GPT-4o + NL2LTL	55.4	91.5	98.4	0.77
GPT-4o + NL2SPEC	56.3	91.9	98.5	0.78
GPT-4o + NL2TL	65.2	94.6	98.1	0.85
GPT-4o + Req2LTL	**88.4**	**100.0**	**99.5**	**0.96**
DeepSeek-V3 + Zero-Shot	34.8	87.5	97.5	0.73
DeepSeek-V3 + NL2LTL	54.0	91.2	98.6	0.78
DeepSeek-V3 + NL2SPEC	54.5	91.9	98.7	0.79
DeepSeek-V3 + NL2TL	58.0	92.9	99.1	0.82
DeepSeek-V3 + Req2LTL	86.6	**100.0**	99.2	**0.96**
Ablation Study on Req2LTL				
D Full Version	**88.4**	**100.0**	**99.5**	**0.96**
A w/o Structured OnionL	65.2	98.2	98.9	0.86
B w/o Stage-wise Decomposition	58.9	94.6	98.1	0.85
C w/o Verification Feedback	84.8	90.2	**99.5**	0.91

TABLE IV
STRUCTURAL COMPLEXITY COMPARISON: INDUSTRIAL VS. ACADEMIC

Metric	Industrial	Academic
Avg. sentence length (tokens)	43.7	15.3
With $\geq$2-layer nesting	63.2%	9.8%
Avg. number of APs	3.7	2.1

semantics. While NL2SPEC, NL2LTL, and NL2TL improve accuracy to the 55–65% range, they still struggle with nested logic, implicit temporal cues, and domain-specific phrasing. Despite enhancements, their syntax validation remains below perfect (<95%), while Req2LTL maintains superior syntax validity at 100%, credited to its rule-based translation engine, which ensures structural correctness by construction. Moreover, its high AP Recall (99.5%) confirms that atomic propositions are effectively extracted, while the gap between recall and exact match reflects the model's ability to preserve both semantics and structure.

The answer to RQ2: Req2LTL achieves best performance on complex industrial requirements, outperforming state-of-the-art methods by a wide margin. Utilizing a hierarchical *OnionL* intermediate representation, it adeptly addresses significant obstacles like temporal ambiguity, logical nesting, and error propagation. The integration of prompt-guided semantic decomposition and deterministic synthesis results in generating high-fidelity LTL formulas with complete syntactic accuracy.

D. RQ3. Ablation Study

To assess the individual contributions of key components within the Req2LTL framework, we conduct an ablation study using GPT-4o as the backend model. Specifically, we compare four configurations: **D** (Full version): Includes structured modeling *OnionL*, staged prompting, and verification feedback. **A** (No Structured Modeling): Bypasses OnionL construction; the LLM generates LTL directly from raw natural language. **B** (No Staged Prompting): Uses OnionL but removes step-wise construction, replacing it with a single monolithic prompt. **C** (No Verification Feedback): Retains modeling and prompting, but skips structural review before translation.

As shown in Table III, the full version (D) consistently outperforms all ablated variants. Removing structured modeling (A) causes semantic accuracy to drop from 88.4% to 65.2%, and BLEU score decreases by more than 10 points. Eliminating staged prompting (B) leads to the lowest semantic accuracy (58.9%), showing that incremental construction is essential for preserving logical structure. Removing verification feedback (C) primarily affects syntactic validity, which drops by nearly 10%, suggesting its role in ensuring output correctness. These results indicate that the observed $> 20\%$ performance gain (from Configuration A to the full version D) stems from the combined effect of structured modeling via *OnionL* and stage-wise decomposition, rather than from OnionL alone. Removing either component leads to significant degradation, showing that both are indispensable and mutually reinforcing.

The answer to RQ3: Each core component of the Req2LTL pipeline contributes substantially to final performance. Structured intermediate representation (*OnionL*) captures semantic hierarchy, staged prompting stabilizes generation, and verification feedback ensures correctness. Together, these modules enable Req2LTL to outperform existing approaches and deliver high-quality translations.

E. RQ4. Structural Complexity and Error Analysis

To investigate whether structural challenges in industrial requirements are systematic and whether Req2LTL mitigates them, we conduct both quantitative and qualitative analyses. Table IV compares our industrial aerospace dataset with standard academic benchmarks. Industrial requirements are significantly more complex across all measured dimensions: they are longer on average, more deeply nested, and reference more atomic propositions than academic samples. These features substantially increase both syntactic complexity and semantic ambiguity, posing challenges for LLMs.

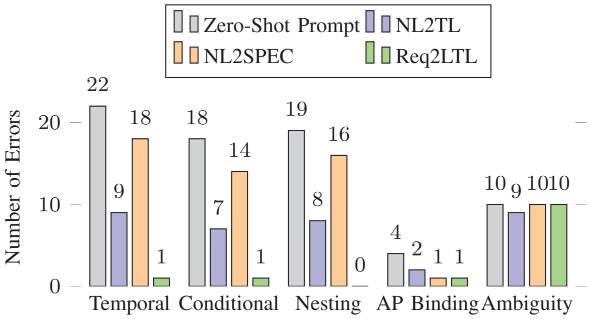


Fig. 5. Error distribution across five categories for four methods. REQ2LTL significantly reduces structural errors, with remaining issues primarily due to ambiguity or contextual omissions.

To investigate how these complexities impact model performance, we classify all incorrect LTL outputs into five categories of structural errors: (1) Temporal Misinterpretation: Failure to capture correct temporal intent, such as precedence or sustained conditions. (2) Conditional Confusion: Logical inversion or misinterpretation of constructs like *if* and *unless*. (3) Loss of Nesting: Flattening of multi-layer logical structures, losing semantic hierarchy. (4) Incorrect AP Binding: Errors in identifying variables, thresholds, or assignments. (5) Ambiguity or Context Omission: Vague or underspecified input leading to multiple plausible interpretations.

Figure 5 shows that Zero-Shot Prompt and NL2TL suffer most from temporal misinterpretation and loss of nesting, each accounting for nearly half of their total errors. This demonstrates the limitations of structure-unaware methods in handling deep logic. NL2SPEC offers modest improvements but still exhibits instability across categories. In contrast, REQ2LTL effectively eliminates errors in the first four categories. Its only remaining issues are due to inherent ambiguity in the input, which is difficult to resolve without contextual grounding.

In summary, compared with existing prompting-based methods, REQ2LTL demonstrates clear advantages on classes of requirements that are particularly challenging: (1) implicit temporal semantics (e.g., “will be set”, “unless”), (2) deeply nested logic structures (63.2% of industrial dataset), (3) pattern-triggered constraints that require correct scope and causality, and (4) long clause dependencies that often exceed the syntactic modeling capacity of end-to-end LLMs. By enforcing structured semantic decomposition, REQ2LTL progressively translates complex sentences and effectively avoids these errors.

The answer to RQ4: Structural challenges are prevalent in real-world industrial requirements, and REQ2LTL effectively mitigates them by enforcing semantic structure and reducing error propagation during generation. Nonetheless, certain cases of linguistic ambiguity persist, motivating the integration of visualized feedback and human-in-the-loop correction, which we explore in the next section.

F. Threats to Validity

One potential threat is data leakage in publicly available academic benchmarks, which may have been seen during pretraining by LLMs. However, our industrial dataset, composed of proprietary aerospace requirements, was never part of any model's training corpus, ensuring unbiased evaluation. Another limitation lies in domain scope: while results are strong in general scenarios and the aerospace field, generalization to other domains is untested. Finally, the quality of all baselines and REQ2LTL depends on the behavior of the underlying LLM, which may vary across model versions or providers. In our experiment, we conducted a fairness treatment.

VII. CASE STUDY AND DISCUSS

A. Representative Failure Cases

While REQ2LTL performs well overall, certain industrial requirements continue to pose challenges. The most common failure type we observe is *ambiguity or context omission*, where the intended timing of actions is not explicitly stated, making precise formalization unreliable.

Example, Req04: *“The control system should as soon as possible initiate the heading adjustment function upon receiving a verified ARINC 429 waypoint command, ultimately reducing the deviation angle to less than 2 degrees.”*

This requirement specifies two subgoals: triggering the heading adjustment promptly and eventually reducing the deviation. However, the phrase “*as soon as possible*” is vague. The model assigns both subgoals the F (eventually) operator:

$$\text{LTL04}: G(\text{WaypointCmd} = \text{True} \rightarrow (F\,(\text{HeadingFun} = \text{True}) \wedge F\,(\text{DevAngleLow} < 2)))$$

The first sub-goal should instead use X, indicating that the action is to be executed at the next time point. The misclassification stems from the natural language's lack of explicit timing, which the model cannot resolve without domain-specific understanding.

Discussion: This case highlights a common challenge in industrial requirement translation: natural language descriptions may contain inherent ambiguity or omit critical information. For instance, phrases like *as soon as possible* suggest urgency but lack formal definition. Without clear temporal anchors or domain-specific context, even structure-aware models tend to default to loose eventualities such as F, resulting in misaligned formalizations. More broadly, this reflects a limitation in the requirement itself. Many industrial specifications rely on shared engineering knowledge and are written with implicit assumptions, rather than precise, machine-interpretable semantics. In such cases, automated systems struggle to infer intent or resolve underspecified constructs.

This inherent ambiguity cannot be fully resolved by automated translation alone, even when guided by structured prompts or intermediate representations. This underscores the need for human-in-the-loop mechanisms. REQ2LTL addresses this by exposing editable intermediate structures through *OnionL*, allowing engineers to inspect, refine, and correct semantic errors. In the next section, we show how this feedback loop enables low-effort repair of misclassifications while retaining automation efficiency.

B. Structured Feedback via Visualized OnionL

To facilitate error correction in cases of semantic ambiguity, REQ2LTL provides a visualized *OnionL* representation to support user inspection and guided refinement. As illustrated in Figure 4, the model's intermediate output is rendered as a structured diagram, enabling engineers to directly examine how a requirement has been semantically interpreted. When inconsistencies are identified, users may either revise the temporal operators within the diagram or provide natural-language feedback to regenerate a corrected structure. In the example Req04 and LTL04, replacing the first F with X in the tree was sufficient to recover the correct semantics, producing the following LTL formula. This correction process is both lightweight and non-intrusive, requiring no modifications to the original input and preserving the automation pipeline while allowing precise semantic control.

$$\text{LTL04: } G(\text{ WaypointCmd} = \text{True} \rightarrow \\ (X\ (\text{HeadingFun} = \text{True}) \land F\ (\text{DevAngleLow} < 2)))$$

Discussion: The visualized *OnionL* structure is not merely an inspection aid—it serves as a pivotal component for enabling human-in-the-loop correction. Users can directly manipulate the tree or interact with the underlying LLM to generate revised outputs, forming a feedback loop that enhances both robustness and interpretability. This mechanism ensures traceability, supports auditability, and aligns with certification workflows, making REQ2LTL a practical tool for safety-critical engineering contexts. Its effectiveness was confirmed during expert evaluation: among 112 industrial requirements, 13 LTL formulas initially exhibited semantic errors. All were corrected within 10 minutes using the visual interface, demonstrating the practicality of the approach in real-world formalization scenarios. It should be emphasized that these 13 manually corrected cases were used solely to demonstrate the practicality of the visualization-based human-in-the-loop interface. All quantitative results reported in RQ1 and RQ2 were obtained from the fully automated pipeline with manual validation disabled, ensuring that the performance metrics strictly reflect automated capability.

VIII. Related Works

Early approaches [30]–[35] for translating natural language (NL) requirements into formal specification relied heavily on handcrafted rules, such as syntactic preprocessing, pattern matching, and attribute grammars. While effective in constrained domains, these methods lack scalability and robustness due to their domain-specific assumptions and limited expressiveness. The advent of neural models introduced data-driven paradigms. Seq2Seq architectures [36], semantic parsers [25], [26], and template-guided generators enabled automatic learning from paired NL–LTL datasets. However, these models often generalize poorly to real-world requirements that exhibit implicit semantics, nested logic, and domain-specific terminology, especially when training data lacks such complexity.

Transformer-based language models [37] such as GPT [38], T5 [39], and PaLM [40] brought significant improvements in text generation [41] and code synthesis [42]. Building on these capabilities, recent methods [16], [23], [24] have explored few-shot prompting, abstraction templates, and fine-tuning for formal specification generation. More recent work [43]–[45] incorporates interactive feedback and decomposition mechanisms to enhance coverage and interpretability across a broader range of scenarios. Nonetheless, LLM-based methods remain sensitive to prompt phrasing and often lack structural transparency. They struggle with faithfully modeling compositional semantics and handling ambiguity in complex industrial requirements. These challenges point to the need for more robust, structure-aware translation frameworks capable of preserving semantic fidelity under domain-specific constraints.

IX. Conclusion

This paper introduced REQ2LTL, a modular framework for translating natural language requirements into LTL specifications. The framework introduces a novel intermediate representation *OnionL*, which decomposes requirements into a compositional tree composed of semantic scopes, logical relations, and atomic propositions, thereby enabling structured and verifiable translation. By combining LLMs for hierarchical semantic decomposition with a deterministic, rule-based translator for validation and LTL conversion, REQ2LTL achieves 88.4% semantic accuracy and 100% syntactic correctness on real-world aerospace requirements, substantially outperforming prior approaches. Our results demonstrate that REQ2LTL effectively bridges the gap between informal requirements and formal specifications, and provides a practical foundation for scalable adoption in safety-critical industrial systems. In future work, we plan to extend the framework to more expressive temporal logics such as STL and MTL.

X. Data Availability

Due to confidentiality restrictions related to aerospace data and the ongoing integration of the tool into our industrial partner's internal verification platform, the full source code and industrial dataset cannot be released at this stage. With partner's approval, we plan to release a standalone prototype of REQ2LTL, together with portions of the aerospace dataset and an interactive front-end interface, once the integration work is complete. In the meantime, a demonstration package of REQ2LTL, including a usage guide and video, is publicly available at https://github.com/Meng-Nan-MZ/Req2LTL.git.

Acknowledgements

This work was supported in part by the National Natural Science Foundation of China (Nos. 62192730, 62192734, 62302375, 62472339, 6240073166), the China Postdoctoral Science Foundation funded project (No. 2023M723736), the Outstanding Youth Science Foundation of Shaanxi Province under Grant 2025JC-JCQN-083, the Fundamental Research Funds for the Central Universities (No. XJSJ24086), and the CCF-Huawei Populus Grove Fund (No. CCF-HuaweiFM202507).

References

[1] S. Paul, E. Cruz, A. Dutta, A. Bhaumik, E. Blasch, G. Agha, S. Patterson, F. Kopsaftopoulos, and C. Varela, "Formal verification of safety-critical aerospace systems," *IEEE Aerospace and Electronic Systems Magazine*, vol. 38, no. 5, pp. 72–88, 2023.

[2] Z. Ma, L. Qiao, M.-F. Yang, S.-F. Li, and J.-K. Zhang, "Verification of real time operating system exception management based on sparcv8," *Journal of Computer Science and Technology*, vol. 36, no. 6, pp. 1367–1387, 2021. [Online]. Available: https://jcst.ict.ac.cn/en/article/doi/10.1007/s11390-021-1644-x

[3] R. Gu, Z. Shao, H. Chen, X. Wu, J. Kim, V. Sjöberg, and D. Costanzo, "Certikos: An extensible architecture for building certified concurrent os kernels," in *Proceedings of the 12th USENIX Symposium on Operating Systems Design and Implementation*. Savannah, GA, USA: USENIX Association, 2016, pp. 653–669. [Online]. Available: https://www.usenix.org/conference/osdi16/technical-sessions/presentation/gu

[4] X. Leroy, S. Blazy, D. Kästner, B. Schommer, M. Pister, and C. Ferdinand, "Compcert-a formally verified optimizing compiler," in *Proceeding of 8th Embedded Real Time Software and Systems, European Congress*, Toulouse, France, 2016. [Online]. Available: https://inria.hal.science/hal-01238879/document

[5] J. D. Backes, S. Bayless, B. Cook, C. Dodge, A. Gacek, A. J. Hu, T. Kahsai, B. Kocik, E. Kotelnikov, J. Kukovec, S. McLaughlin, J. Reed, N. Rungta, J. Sizemore, M. A. Stalzer, P. Srinivasan, P. Subotić, C. Varming, and B. Whaley, "Reachability analysis for aws-based networks," in *Proceedings of the 31st International Conference on Computer Aided Verification*, ser. Lecture Notes in Computer Science, vol. 11562. New York City, NY, USA: Springer, 2019, pp. 231–241. [Online]. Available: https://doi.org/10.1007/978-3-030-25543-5_14

[6] A. Cimatti, E. M. Clarke, E. Giunchiglia, F. Giunchiglia, M. Pistore, M. Roveri, R. Sebastiani, and A. Tacchella, "Nusmv 2: An opensource tool for symbolic model checking," in *International Conference on Computer Aided Verification*, 2002. [Online]. Available: https://api.semanticscholar.org/CorpusID:138242

[7] A. Duret-Lutz, A. Lewkowicz, A. Fauchille, T. Michaud, E. Renault, and L. Xu, "Spot 2 . 0 — a framework for ltl and ω-automata manipulation," 2016. [Online]. Available: https://api.semanticscholar.org/CorpusID:53473387

[8] S. Phipathananunth, "Using mutations to analyze formal specifications," in *Proceeding of the 37th Companion Proceedings of the International Conference on Systems, Programming, Languages, and Applications: Software for Humanity*. Auckland, New Zealand: ACM, 2022, pp. 81–83. [Online]. Available: https://doi.org/10.1145/3563768.3563960

[9] A. Liu and S. Liu, "Enhancing the capability of testing-based formal verification by handling operations in software packages," *IEEE Trans. Software Eng.*, vol. 49, no. 1, pp. 304–324, 2023. [Online]. Available: https://doi.org/10.1109/TSE.2022.3150333

[10] A. Mashkoor, M. Leuschel, and A. Egyed, "Validation obligations: A novel approach to check compliance between requirements and their formal specification," *Proceeding of the 43rd IEEE/ACM International Conference on Software Engineering: New Ideas and Emerging Results*, pp. 1–5, 2021. [Online]. Available: https://doi.org/10.1109/ICSE-NIER52604.2021.00009

[11] C. Wen, J. Cao, J. Su, Z. Xu, S. Qin, M. He, H. Li, S. Cheung, and C. Tian, "Enchanting program specification synthesis by large language models using static analysis and program verification," pp. 302–328, 2024.

[12] A. Blasi, A. Goffi, K. Kuznetsov, A. Gorla, M. D. Ernst, M. Pezzè, and S. D. Castellanos, "Translating code comments to procedure specifications," *Proceedings of the 27th ACM SIGSOFT International Symposium on Software Testing and Analysis*, 2018. [Online]. Available: https://api.semanticscholar.org/CorpusID:49863340

[13] S. H. Tan, D. Marinov, L. Tan, and G. T. Leavens, "@tcomment: Testing javadoc comments to detect comment-code inconsistencies," *2012 IEEE Fifth International Conference on Software Testing, Verification and Validation*, pp. 260–269, 2012. [Online]. Available: https://api.semanticscholar.org/CorpusID:11189276

[14] J. Pan, G. Chou, and D. Berenson, "Data-efficient learning of natural language to linear temporal logic translators for robot task specification," in *Proceedings of the 40th IEEE International Conference on Robotics and Automation*. London, UK: IEEE, 2023, pp. 11 554–11 561. [Online]. Available: https://doi.org/10.1109/ICRA48891.2023.10161125

[15] S. Rongali, K. Arkoudas, M. Rubino, and W. Hamza, "Training naturalized semantic parsers with very little data," in *Proceedings of the 31st International Joint Conference on Artificial Intelligence*. Vienna, Austria: ijcai.org, 2022, pp. 4353–4359. [Online]. Available: https://doi.org/10.24963/ijcai.2022/604

[16] Y. Chen, R. Gandhi, Y. Zhang, and C. Fan, "Nl2tl: Transforming natural languages to temporal logics using large language models," in *Proceedings of the Conference on Empirical Methods in Natural Language Processing*. Singapore: Association for Computational Linguistics, 2023, pp. 15 880–15 903. [Online]. Available: https://doi.org/10.18653/v1/2023.emnlp-main.985

[17] OpenAI, "Gpt-4o technical report," https://openai.com/index/gpt-4o, 2024, accessed: May 2025.

[18] E. Nijkamp, B. Pang, H. Hayashi, L. Tu, H. Wang, Y. Zhou, S. Savarese, and C. Xiong, "Codegen: An open large language model for code with multi-turn program synthesis," in *International Conference on Learning Representations*, 2022. [Online]. Available: https://api.semanticscholar.org/CorpusID:252668917

[19] C. Wen, Y. Cai, B. Zhang, J. Su, Z. Xu, D. Liu, S. Qin, Z. Ming, and T. Cong, "Automatically inspecting thousands of static bug warnings with large language model: How far are we?" *ACM Transactions on Knowledge Discovery from Data*, vol. 18, no. 7, pp. 1–34, 2024.

[20] J. Cao, Y. Lu, M. Li, H. Ma, H. Li, M. He, C. Wen, L. Sun, H. Zhang, S. Qin *et al.*, "From informal to formal–incorporating and evaluating llms on natural language requirements to verifiable formal proofs," *arXiv preprint arXiv:2501.16207*, 2025.

[21] X. Du, M. Liu, K. Wang, H. Wang, J. Liu, Y. Chen, J. Feng, C. Sha, X. Peng, and Y. Lou, "Evaluating large language models in class-level code generation," in *Proceedings of the IEEE/ACM 46th International Conference on Software Engineering*, 2024, pp. 1–13.

[22] J. Su, L. Deng, C. Wen, S. Qin, and C. Tian, "Cfstra: Enhancing configurable program analysis through llm-driven strategy selection based on code features," in *International Symposium on Theoretical Aspects of Software Engineering*. Springer, 2024, pp. 374–391.

[23] M. Cosler, C. Hahn, D. Mendoza, F. Schmitt, and C. Trippel, "nl2spec: Interactively translating unstructured natural language to temporal logics with large language models," in *Proceedings of the 35th International Conference on Computer Aided Verification*, ser. Lecture Notes in Computer Science, vol. 13965. Paris, France: Springer, 2023, pp. 383–396. [Online]. Available: https://doi.org/10.1007/978-3-031-37703-7_18

[24] F. Fuggitti and T. Chakraborti, "Nl2ltl - a python package for converting natural language (nl) instructions to linear temporal logic (ltl) formulas," in *Proceedings of the 37th AAAI Conference on Artificial Intelligence, the 35th Conference on Innovative Applications of Artificial Intelligence, the 13th Symposium on Educational Advances in Artificial Intelligence*. Washington, DC, USA: AAAI Press, 2023, pp. 16 428–16 430. [Online]. Available: https://doi.org/10.1609/aaai.v37i13.27068

[25] J. He, E. Bartocci, D. Nickovic, H. Isakovic, and R. Grosu, "Deepstl - from english requirements to signal temporal logic," pp. 610–622, 2022. [Online]. Available: https://doi.org/10.1145/3510003.3510171

[26] C. Wang, C. Ross, B. Katz, and A. Barbu, "Learning a natural-language to ltl executable semantic parser for grounded robotics," in *Proceedings of the 4th Conference on Robot Learning*, ser. Proceedings of Machine Learning Research, vol. 155. Virtual Event / Cambridge, MA, USA: PMLR, 2020, pp. 1706–1718. [Online]. Available: https://proceedings.mlr.press/v155/wang21g.html

[27] K. M. Knatten and contributors, "Mermaid - generation of diagrams and flowcharts from text in a similar manner as markdown," https://mermaid.js.org/, 2024, version 10.6.1, Accessed: 2025-05-22.

[28] A. Cimatti, E. Clarke, F. Giunchiglia, M. Roveri, A. Tacchella, and R. Sebastiani, "Nusmv: a new symbolic model checker," *International Journal on Software Tools for Technology Transfer (STTT)*, vol. 2, pp. 410–425, 2000.

[29] S. Zhang, H. Zhao *et al.*, "Deepseek-vl: Scaling vision-language models with vision token learner," *arXiv preprint arXiv:2405.07927*, 2024. [Online]. Available: https://arxiv.org/abs/2405.07927

[30] B. Swick, "Flexible robot programming through human-guided state machine synthesis with large language models," Ph.D. dissertation, ProQuest Dissertations Publishing, 2024. [Online]. Available: https://search.proquest.com/openview/d424c3727a1721b3af11244d1077e54e/1

[31] S. Tellex, T. Kollar, S. Dickerson, M. R. Walter, A. G. Banerjee, S. J. Teller, and N. Roy, "Approaching the symbol grounding problem with probabilistic graphical models," *AI Mag.*, vol. 32, no. 4, pp. 64–76, 2011. [Online]. Available: https://doi.org/10.1609/aimag.v32i4.2384

[32] Y. Xu, J. Feng, and W. Miao, "Learning from failures: Translation of natural language requirements into linear temporal logic with large language models," in *Proceedings of the 2024 IEEE 24th International Conference on Software Engineering and Formal Methods (SEFM)*. IEEE, 2024. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/10684640

[33] I. Buzhinsky, "Formalization of natural language requirements into temporal logics: A survey," in *2019 IEEE 17th International Conference on Industrial Informatics (INDIN)*. IEEE, 2019. [Online]. Available: https://www.researchgate.net/publication/334635667

[34] N. Wu, Y. Li, H. Yang, H. Chen, S. Dai, and C. Hao, "Survey of machine learning for software-assisted hardware design verification: Past, present, and prospect," *ACM Transactions on Design Automation of Electronic Systems*, 2024. [Online]. Available: https://dl.acm.org/doi/abs/10.1145/3661308

[35] S. Zhang, J. Zhai, L. Bu, M. Chen, L. Wang, and X. Li, "Automated generation of ltl specifications for smart home iot using natural language," *2020 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pp. 622–625, 2020. [Online]. Available: https://api.semanticscholar.org/CorpusID:219854669

[36] N. Gopalan, D. Arumugam, L. L. S. Wong, and S. Tellex, "Sequence-to-sequence language grounding of non-markovian task specifications," in *Proceedings of the Robotics: Science and Systems XIV, Carnegie Mellon University*, Pittsburgh, Pennsylvania, USA, 2018. [Online]. Available: http://www.roboticsproceedings.org/rss14/p67.html

[37] A. Vaswani, N. M. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, "Attention is all you need," in *Proceedings of the Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems*, Long Beach, CA, USA, 2017, pp. 5998–6008. [Online]. Available: https://proceedings.neurips.cc/paper/2017/hash/3f5ee243547dee91fbd053c1c4a845aa-Abstract.html

[38] A. Madotto and Z. Liu, "Language models as few-shot learner for task-oriented dialogue systems," *CoRR*, vol. abs/2008.06239, 2020. [Online]. Available: https://arxiv.org/abs/2008.06239

[39] C. Raffel, N. M. Shazeer, A. Roberts, K. Lee, S. Narang, M. Matena, Y. Zhou, W. Li, and P. J. Liu, "Exploring the limits of transfer learning with a unified text-to-text transformer," *J. Mach. Learn. Res.*, vol. 21, pp. 140:1–140:67, 2020. [Online]. Available: http://jmlr.org/papers/v21/20-074.html

[40] A. Chowdhery, S. Narang, J. Devlin, M. Bosma, G. Mishra, A. Roberts, P. Barham, H. W. Chung, C. Sutton, S. Gehrmann, P. Schuh, K. Shi, S. Tsvyashchenko, J. Maynez, A. Rao, P. Barnes, Y. Tay, N. M. Shazeer, V. Prabhakaran, E. Reif, N. Du, B. C. Hutchinson, R. Pope, J. Bradbury, J. Austin, M. Isard, G. Gur-Ari, P. Yin, T. Duke, A. Levskaya, S. Ghemawat, S. Dev, H. Michalewski, X. García, V. Misra, K. Robinson, L. Fedus, D. Zhou, D. Ippolito, D. Luan, H. Lim, B. Zoph, A. Spiridonov, R. Sepassi, D. Dohan, S. Agrawal, M. Omernick, A. M. Dai, T. S. Pillai, M. Pellat, A. Lewkowycz, E. Moreira, R. Child, O. Polozov, K. Lee, Z. Zhou, X. Wang, B. Saeta, M. Díaz, O. Firat, M. Catasta, J. Wei, K. S. Meier-Hellstern, D. Eck, J. Dean, S. Petrov, and N. Fiedel, "Palm: Scaling language modeling with pathways," *J. Mach. Learn. Res.*, vol. 24, pp. 240:1–240:113, 2022. [Online]. Available: http://jmlr.org/papers/v24/22-1144.html

[41] B. Chintagunta, N. Katariya, X. Amatriain, and A. Kannan, "Medically aware gpt-3 as a data generator for medical dialogue summarization," in *Proceedings of the Machine Learning for Healthcare Conference*, ser. Proceedings of Machine Learning Research, vol. 149. Virtual Event: PMLR, 2021, pp. 354–372. [Online]. Available: https://proceedings.mlr.press/v149/chintagunta21a.html

[42] M. Chen, J. Tworek, H. Jun, Q. Yuan *et al.*, "Evaluating large language models trained on code," *CoRR*, vol. abs/2107.03374, 2021. [Online]. Available: https://arxiv.org/abs/2107.03374

[43] J. Wang, D. Sundarsingh, and J. Deshmukh, "Conformalnl2ltl: Translating natural language instructions into temporal logic formulas with conformal correctness guarantees," *arXiv preprint arXiv:2504.21022*, 2025. [Online]. Available: https://arxiv.org/abs/2504.21022

[44] M. Zhao, R. Tao, Y. Huang, J. Shi, S. Qin, and Y. Yang, "Nl2ctl: Automatic generation of formal requirements specifications via large language models," in *International Conference on Formal Engineering Methods*. Springer, 2024, pp. 1–17.

[45] D. Mendoza, C. Hahn, and C. Trippel, "Translating natural language to temporal logics with large language models and model checkers," *2024 Formal Methods in Computer-Aided Design (FMCAD)*, pp. 1–11, 2024. [Online]. Available: https://api.semanticscholar.org/CorpusID:273159395