A Retrieval-Augmented Generation System for Automated Functional Safety Analysis of AUTOSAR Basic Software Module Dependencies

Mohand Hammad[1]*, Ahmed Moro[2], Mohamed Taher[3]
Faculty of Engineering, Department of Computer and Systems Engineering,
Ain Shams University, Cairo 11517, Egypt[1,3]
Siemens Digital Industries Software, Lifecycle Collaboration Software Segment, Cairo 11835, Egypt[1,2]

***Abstract*—This study presents an advanced Retrieval-Augmented Generation (RAG) system designed to assist functional safety engineers in performing safety analysis of AUTOSAR Classic Platform Basic Software (BSW) module dependencies. The system extracts structured dependency information from 128 AUTOSAR Software Specification (SWS) documents in ARXML format and generates draft Failure Mode and Effects Analysis (FMEA), Fault Tree Analysis (FTA), and Dependent Failure Analysis (DFA) tables compliant with AIAG VDA, IEC 60812, IEC 61025, and ISO 26262 standards for human expert review and approval. Key innovations include: 1) LLM-driven table definition extraction that designs optimal analysis output formats based on merged AUTOSAR safety context, ISO 26262 lifecycle considerations, and standard methodologies; 2) content-based inter-module dependency validation that prevents hallucination of non-existent module interactions; 3) ASIL-aware analysis that prioritizes lower-integrity components corrupting higher-integrity components per ISO 26262 freedom from interference; 4) a modular architecture with dual interfaces (CLI tool and LangGraph-based conversational chatbot) where the chatbot reuses core RAG functions, enabling single-source maintenance. The architecture combines semantic chunking with metadata-based filtering for precise module retrieval, episodic and working memory for multi-turn sessions, and automated Excel report generation with source traceability. A comparative evaluation against an LLM-only baseline and a standard semantic-search RAG baseline demonstrates that metadata filtering with content validation eliminates hallucinated dependencies. On a curated stress-test dataset of 15 safety-critical modules representing the most complex BSW interdependencies (watchdog supervision, diagnostics, memory management, communication stacks), the system achieves perfect micro-averaged precision/recall across 95 documented dependencies. Preliminary expert validation by three functional safety engineers confirmed the practical utility of the generated analyses as draft starting points for formal safety assessments.**

***Keywords*—AUTOSAR; RAG; Retrieval-Augmented Generation; functional safety; FMEA; FTA; DFA; ISO 26262; Langhain; vector database; automotive software; safety analysis**

I. Introduction

The increasing complexity of automotive electrical and electronic (E/E) architectures has driven the adoption of standardized software platforms, with AUTOSAR (AUTomotive Open System ARchitecture) emerging as the dominant framework for Basic Software (BSW) development in the automotive industry [1]. The AUTOSAR Classic Platform R25-11 release defines 128 BSW modules with intricate inter-module dependencies spanning communication, diagnostics, memory, supervision, and operating system layers [2]. ISO 26262 [3] mandates rigorous safety analyses—Failure Mode and Effects Analysis (FMEA), Fault Tree Analysis (FTA), and Dependent Failure Analysis (DFA)—to identify hazards arising from these dependencies. Industry experience suggests that a single FMEA for one BSW module pair typically requires 4–8 engineering hours of manual specification review, cross-referencing, and table construction; scaling this effort across all 128 modules and their combinatorial dependency paths remains a prohibitive bottleneck for many development organizations.

A. Gaps in Existing Work

Despite growing research on LLM-based safety engineering (Section II), three critical gaps persist. First, existing RAG systems for AUTOSAR [4], [5] target configuration validation or general Q&A rather than structured safety analysis artifact generation. Second, LLM-based FMEA research [6], [7] operates on customer reviews or system-level sensor data rather than specification-level module dependencies with requirement traceability. Third, no prior work addresses *hallucination prevention* in safety analysis contexts, where a single fabricated dependency can trigger costly and unnecessary design reviews or, worse, mask real hazards.

B. Contributions

This study presents a RAG system that bridges these gaps by generating draft FMEA/FTA/DFA tables grounded in AUTOSAR SWS specifications for human expert review and approval (human-in-the-loop per ISO 26262 accountability). The system comprises a Python CLI (`rag_query.py`) and a conversational chatbot (`chatbot.py`) sharing core logic. The principal contributions are:

- Content-based dependency validation that prevents hallucination of non-existent module interactions by verifying cross-module API references, header inclusions, and interface documentation in retrieved ARXML specifications before generating analysis.
- Metadata-filtered retrieval via Qdrant that achieves exact module scoping, eliminating semantic drift be-

*Corresponding author.

tween similarly named modules (e.g., Can vs. CanIf vs. CanSM).

- ARXML extraction pipeline targeting scheduled functions, expected interfaces, and dependency sections from 128 SWS documents with full requirement traceability (SWS/SRS identifiers).
- LLM-driven table definition extraction that designs standards-compliant output formats by merging AUTOSAR fault models with AIAG VDA [8], IEC 60812 [9], IEC 61025 [10], and ISO 26262-9 [11] methodologies.
- Comparative evaluation against an LLM-only baseline and a standard semantic-search RAG baseline, with ablation experiments quantifying the contribution of metadata filtering and content validation.
- Robustness evaluation on adversarial queries (unrelated module pairs, ambiguous names, non-existent modules) to measure hallucination risk under realistic conditions.
- Preliminary expert validation by three functional safety engineers who confirmed practical utility of generated analyses as starting points for formal assessments.
- Dual-interface modular architecture with automated Excel report generation supporting audit compliance and source traceability.

The remainder of this study is organized as follows: Section II provides background on AUTOSAR, functional safety standards, and RAG architectures. Section III details the system architecture and methodology. Section IV describes the implementation. Section V presents experimental results including baseline comparisons, ablation studies, and robustness evaluation. Section VI discusses findings, expert feedback, and limitations. Section VII concludes with future work directions.

II. Background and Related Work

A. AUTOSAR Classic Platform Architecture

AUTOSAR CP defines layered architecture for ECUs: BSW (MCAL, ECU Abstraction, Services, Complex Drivers), RTE, and Application [1]. The R25-11 release contains 128 SWS documents defining module APIs, configuration, scheduled functions, expected interfaces, and traceability.

B. Functional Safety and ISO 26262

ISO 26262 [3] mandates FMEA (bottom-up failure analysis), FTA (top-down deductive analysis), and DFA (common cause/cascading failures). For AUTOSAR, analyses must consider BSW interactions, shared resources, timing dependencies, and safety mechanisms (WdgM, E2E, OS partitioning).

C. Retrieval-Augmented Generation

RAG enhances LLMs by retrieving context from knowledge bases before generation [12], combining parametric memory (pre-trained LLM) with non-parametric memory (dense vector index [13]) for context-aware generation. Recent surveys [14] identify RAG as a key paradigm for grounding transformer-based architectures [15] in domain-specific knowledge. RAG addresses LLM limitations: knowledge expansion, provenance, and hallucinations [16], [17].

D. Related Work

Recent work on functional safety-specific LLM systems (e.g., Aegis [18]) demonstrates multi-agent RAG for HARA, FSR documentation, and test planning. In contrast, our work targets AUTOSAR Classic Platform BSW dependency analysis and standards-driven FMEA/FTA/DFA generation grounded in SWS ARXML specifications. The approaches are complementary: Aegis addresses concept- and system-level artifacts, while our system provides module-level, specification-grounded safety analysis with traceable interface dependencies.

Petrovic et al. [19] propose an LLM-empowered workflow for functional safety and security analysis at the implementation level, using event chains to validate Python code behavior against safety rules in ADAS scenarios. Their approach extracts VSS (Vehicle Signal Specification) and CAN messages from executable code, constructs event chain models in PlantUML, and validates them against OCL constraints using Model-Driven Engineering. While their work focuses on code-level runtime behavior analysis integrated with simulation platforms (CARLA, digital.auto), our system operates at an earlier lifecycle stage, extracting dependency information from AUTOSAR SWS specifications to enable pre-implementation safety analysis. The approaches are complementary—Petrovic's system validates implementation compliance with safety rules, while our system provides specification-level dependency analysis before code is written.

Sevenhuijsen et al. [20] investigate the feasibility of using LLMs to generate formally verified C code for industrial automotive safety-critical modules from Scania, achieving functional correctness through Frama-C verification with ACSL specifications and adherence to ISO 26262 non-functional requirements. Their evaluation of ten LLMs across three embedded control modules demonstrated success rates of 540/800, 59/800, and 46/800 for modules of varying complexity, with closed-source LLMs outperforming open-source alternatives. While their work focuses on generating verified implementation code from formal specifications, our system addresses a complementary challenge—generating safety analysis artifacts (FMEA/FTA/DFA tables) from AUTOSAR SWS specifications. Both approaches leverage LLMs with formal specifications in the ISO 26262 automotive safety domain, but target different lifecycle stages: they generate code that must be verified, while the present system generates safety analysis documentation that guides design decisions.

GenAI-based FTA research has explored prompt-engineered LLMs for autonomy sensor fault trees (e.g., lidar failure cascades) [7], including PlantUML diagram synthesis and qualitative evaluation against public LLMs. This line of work demonstrates feasibility for FTA generation but centers on system-level sensor diagrams rather than specification-grounded, module-level dependency analysis. Our system instead enforces documented API-level constraints from AUTOSAR SWS and produces standards-compliant FMEA/FTA/DFA tables with complete evidence traceability.

Automatic FMEA tools in the model-based design community integrate Simulink models with fault injection and rule-based severity scoring to auto-generate FMEA reports [21]. These approaches rely on executable models and fault-injection campaigns for a specific system design. Our approach complements them by operating directly on AUTOSAR SWS specifications, enabling BSW-wide dependency analysis without requiring a Simulink model and extending beyond FMEA to support FTA and DFA.

Hassani et al. [6] developed an LLM-based framework for automating FMEA generation from automotive customer reviews, achieving 84% accuracy in extracting failure modes and 85.5% relevance for effects using GPT-3.5 Turbo with fine-tuning. Their work demonstrates the feasibility of LLM-driven FMEA in the automotive domain but operates at a different lifecycle stage—extracting user-reported issues from product reviews rather than analyzing specification-level dependencies. While their approach provides valuable post-deployment failure insights from customer feedback, our system targets pre-implementation safety analysis by extracting inter-module dependencies from AUTOSAR SWS specifications to generate standards-compliant FMEA/FTA/DFA tables with complete traceability to source requirements.

Bahr et al. [22] propose a knowledge graph-enhanced RAG (KG-RAG) framework for FMEA in manufacturing contexts, combining Neo4j graph queries with vector search to enable both semantic question-answering and analytical queries on structured FMEA data. Their approach addresses a key limitation of pure vector-based RAG—the loss of symbolic meaning in numerical data—by augmenting the non-parametric store with a knowledge graph that preserves risk priority numbers, severity scores, and occurrence ratings for analytical reasoning. Their user experience study demonstrated 124% improvement in usability and 74% better perceived retrieval time compared to Excel-based FMEA analysis. While their work focuses on manufacturing process FMEA with tabular data sources, our system addresses specification-level FMEA generation from AUTOSAR ARXML documents. The approaches are complementary: their KG-RAG enables efficient retrieval and analysis of existing FMEA data, while our system generates new FMEA tables from specification dependencies with automated severity and occurrence assessment based on safety standards.

Mohamed's RAG-based ARXML validation [5] validates configuration changes for code review (Gerrit workflows), while our system generates safety analyses. Both leverage metadata-enriched retrieval for AUTOSAR's structured format.

Kieu and Bergstrand [4] validated that advanced RAG techniques (metadata filtering, reranking, context compression) significantly improve AUTOSAR specification retrieval quality. The present system adopts their metadata-filtered retrieval principle for module-specific dependency extraction.

III. Methodology

A. System Architecture Overview

The system architecture, illustrated in Fig. 1, comprises three distinct layers that work together to enable automated safety analysis. The **Data Layer** contains AUTOSAR SWS ARXML specifications (128 files from R25-11 release), AUTOSAR explanatory documents providing safety context, and safety analysis standards (IEC/ISO PDFs). The **Processing Layer** includes the ARXML extractor (which parses structured AUTOSAR documents), the standards summarizer (which processes PDF safety analysis methodologies), and an LLM-driven table definition extractor that designs optimal output formats. The **Application Layer** consists of two interfaces: a command-line RAG query tool (`rag_query.py`) and an interactive conversational chatbot (`chatbot.py`), both of which import and reuse core functions from the shared RAG module to ensure consistency.

B. ARXML Document Extraction

AUTOSAR SWS documents are provided in ARXML format, an XML schema defined by the AUTOSAR consortium. Each SWS document follows a hierarchical package structure containing documentation chapters organized by functional areas. The extraction pipeline specifically targets three critical sections for safety analysis:

- Scheduled Functions: Periodic tasks executed by the BSW Scheduler, including MainFunction calls and their timing requirements
- Expected Interfaces: APIs and service interfaces that the module requires from other BSW modules, defining inter-module dependencies
- Dependencies to Other Modules: Explicit textual descriptions of architectural relationships and interface contracts

The extraction process preserves critical information including requirement identifiers (e.g., `SWS_WdgM_00406`) for full traceability to source specifications, API function names extracted from `TT` elements with type annotations, table structures containing detailed interface specifications with parameters and return types, and cross-references to related AUTOSAR specifications. A comprehensive module abbreviation mapping table (e.g., WatchdogManager $\rightarrow$ WdgM, DiagnosticEventManager $\rightarrow$ Dem) enables natural language query resolution from user inputs.

To ensure robustness across variations in AUTOSAR document structure between different BSW modules, the extractor implements several normalization strategies: whitespace normalization and removal of excessive line breaks, merging of contiguous paragraph fragments that represent logically connected content, and flattening of nested list structures into consistent paragraph representations. This normalization reduces downstream LLM prompt variability and ensures that scheduled functions and expected interfaces are represented with stable structural patterns even when chapter nesting levels differ between SWS documents from different AUTOSAR working groups.

C. Vector Database Design

The system employs Qdrant [23] as its vector database, chosen specifically for its robust support of metadata filtering alongside semantic search capabilities. Each document chunk extracted from an ARXML file is stored with a comprehensive metadata schema that enables precise retrieval:

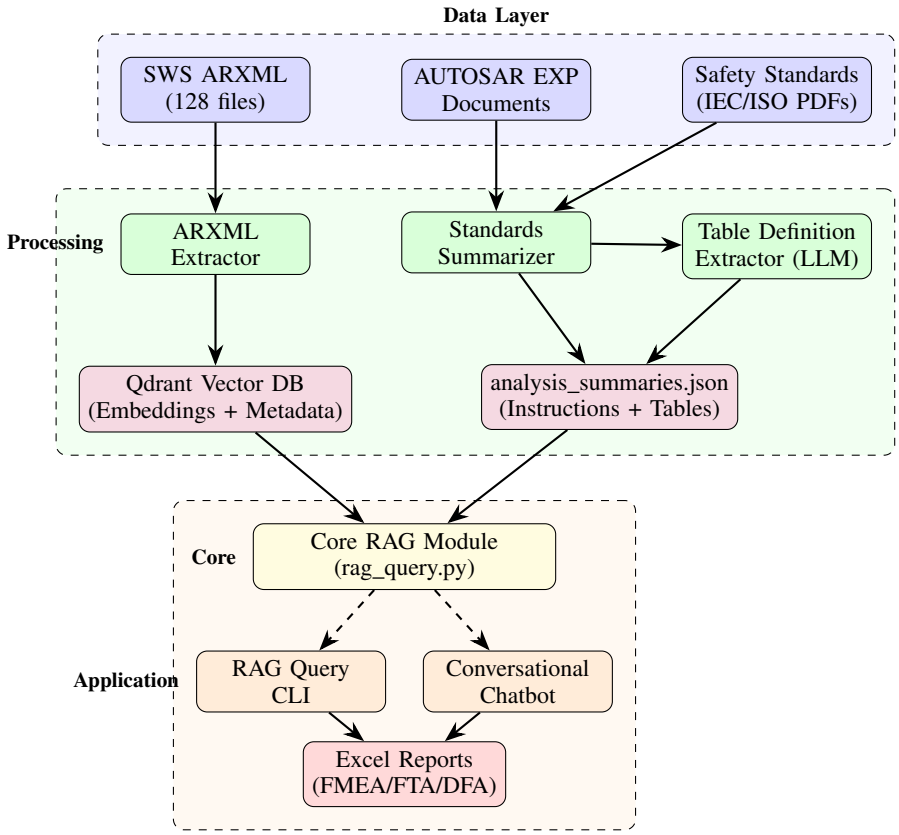

Fig. 1. System architecture of the AUTOSAR Safety Analysis RAG System. The core module (rag_query.py) contains all shared functions; both the CLI and chatbot import from this module (dashed arrows). Table definitions are extracted by an LLM based on merged context from standards and AUTOSAR documents.

- source: Full ARXML filename (e.g., AUTOSAR_CP_SWS_WatchdogManager.arxml) for traceability
- module_name: Full canonical module name (e.g., WatchdogManager, DiagnosticEventManager)
- abbreviation: Standard AUTOSAR module abbreviation (e.g., WdgM, Dem) used for query resolution
- has_scheduled_functions: Boolean flag indicating presence of MainFunction or scheduled task specifications
- has_expected_interfaces: Boolean flag indicating presence of required API dependencies
- has_dependencies_to_other_modules: Boolean flag for explicit inter-module dependency documentation

This metadata-rich design enables two complementary retrieval strategies optimized for different query types:

- Metadata Filtering for Safety Analysis: When users request safety analyses (FMEA/FTA/DFA) for specific modules, the system uses Qdrant's exact-match filtering on the abbreviation field. For example, a query about "WdgM" retrieves only chunks where abbreviation="WdgM", ensuring zero cross-module noise in the analysis context. This approach prioritizes precision over recall, which is essential for safety-critical workflows, where false positive dependencies could trigger unnecessary and expensive design reviews or lead to incorrect safety argumentation.
- Semantic Search for Exploratory Queries: When users pose open-ended questions like "What safety mechanisms does AUTOSAR provide for timing protection?", the system embeds the query using the qwen3-embedding-8b model (1024-dimensional embeddings) and retrieves the top-k most semantically similar chunks across all modules, enabling knowledge discovery and comparative analysis.

Metadata filtering is critical for preventing hallucinated dependencies. Without explicit module scoping, LLMs frequently confuse modules with similar names or related functionality (e.g., CanIf vs. CanTrcv, WdgM vs. WdgIf, Dem vs. Dcm), leading to spurious safety analysis outputs. The metadata layer acts as a hard constraint that filters the document collection before semantic similarity ranking, ensuring that retrieval stays within the specified module scope while allowing semantic search to rank relevant content within that constrained space.

D. Safety Analysis Integration

Safety analysis instructions are carefully integrated from multiple authoritative standards including IEC 60812 (FMEA methodology) [9], IEC 61025 (FTA methodology) [10], and ISO 26262 (automotive functional safety lifecycle) [3]. The

integration follows a multi-stage processing pipeline: 1) text extraction from PDF documents using PyMuPDF to obtain raw methodology descriptions, 2) synthesis of AUTOSAR-specific fault models including memory corruption scenarios (stack overflow, heap corruption), timing violations (deadline misses, race conditions), communication faults (message loss, sequence errors), control flow corruption (unintended function calls, infinite loops), and watchdog timeout scenarios (supervision cycle overruns), and 3) LLM-driven table definition extraction that analyzes the merged context and designs optimal output formats including column names, data types, column widths for Excel formatting, and risk assessment thresholds (severity/occurrence/detection scales).

The generated table definitions are serialized to JSON format in the `analysis_summaries.json` configuration file, ensuring perfect consistency between the analysis generation phase (where the LLM populates table rows) and the Excel export phase (which reads column definitions to format output spreadsheets). This architecture includes built-in column validation: if the LLM attempts to generate analysis rows with columns not present in the table definition, the system raises a validation error rather than producing inconsistent outputs.

E. Conversational Memory Architecture

The conversational chatbot component is architected using LangGraph [24], which provides a state machine framework for complex multi-turn interactions. The system implements three distinct memory types following cognitive science principles:

- Working Memory: Current session state maintained within LangGraph's state graph, including active module names being analyzed, selected analysis type (FMEA/FTA/DFA), FMEA standard variant selection (AIAG VDA vs. IEC 60812), cached retrieval results from previous queries in the same session, extracted table definitions for the current analysis, and retrieved ARXML chunks. This transient memory persists only for the duration of the current interactive session and is cleared when the user exits the chatbot.
- Episodic Memory: Persistent conversation history implemented through LangGraph's MemorySaver checkpointer, which stores previous user messages, LLM-generated responses, intermediate analysis results, and user feedback. This enables the chatbot to reference earlier analyses without re-querying the vector database, supporting natural follow-up questions like "How does this compare to the WdgM analysis performed earlier?" or "Regenerate that FMEA with higher severity thresholds".
- Semantic Memory: The Qdrant vector store serving as the system's long-term knowledge base, containing embedded representations of all 128 AUTOSAR BSW module specifications, safety analysis methodologies from standards documents, and AUTOSAR-specific fault patterns. This memory is persistent across all sessions and shared by all users.

The LangGraph state machine architecture, visualized in Fig. 2, comprises nine specialized nodes with conditional edge routing based on query characteristics:

- understand: Analyzes user intent using LLM-based intent classification to extract target module names, desired analysis type (FMEA/FTA/DFA), and any special constraints (ASIL levels, specific failure modes)
- rewrite: Reformulates ambiguous or incomplete queries for clarity, including expanding abbreviations ("WM" $\rightarrow$ "WdgM"), standardizing terminology ("fault tree" $\rightarrow$ "FTA"), and resolving pronoun references from conversation context
- retrieve: Executes metadata-filtered Qdrant queries to fetch relevant ARXML chunks for the extracted module names, applying boolean filters to ensure only specified modules are retrieved
- analyze: Invokes the LLM with merged safety standards context (AUTOSAR fault models + IEC/ISO methodologies) to generate structured FMEA, FTA, or DFA analyses following the extracted table definitions
- qa_generation: Creates concise question-answer style summaries of analysis results to improve user comprehension, highlighting key failure modes and high-risk interactions
- filter_results: Applies domain-specific validation rules including ASIL level consistency checks (ensuring lower-ASIL modules don't compromise higher-ASIL components), dependency existence verification against documented interfaces, and safety goal alignment with ISO 26262 requirements
- export_excel: Generates formatted Excel workbooks with custom column widths derived from table definitions, freeze panes for header rows, source traceability columns linking to ARXML files, and multi-sheet layouts for complex analyses
- clarify: Routes ambiguous or underspecified queries back to the user for clarification before proceeding, asking targeted questions like "Which FMEA standard would you prefer: AIAG VDA or IEC 60812?"
- respond: Formats the final response with rich context including analysis summaries, file paths to generated Excel reports, links to source AUTOSAR documents, and suggested follow-up actions

Conditional edge routing enables flexible conversation flows: if the query lacks sufficient information or the analysis type cannot be determined with confidence, flow routes to the `clarify` node and loops back to `understand` after receiving user input; if the user explicitly requests Excel export ("save as Excel", "generate report"), flow routes from `filter_results` to `export_excel`; otherwise, flow proceeds directly to `respond` for conversational output. This architecture supports both linear single-shot analyses (one question, one answer) and iterative multi-turn workflows (progressive refinement, comparative analyses, what-if scenarios).

Critically, the chatbot implementation (`chatbot.py`) maintains architectural modularity by importing and reusing all core RAG functions from the base query module (`rag_query.py`), including `get_llm()` for model initialization, `get_qdrant_client()` for vector store connec-

tions, `retrieve_chunks_by_modules()` for metadata-filtered retrieval, `format_context()` for prompt construction, and `generate_excel_report()` for output formatting. This single-source design ensures that bug fixes, API enhancements, and behavioral changes need only be applied once in the core module, automatically propagating to both the command-line interface and the conversational chatbot without code duplication.

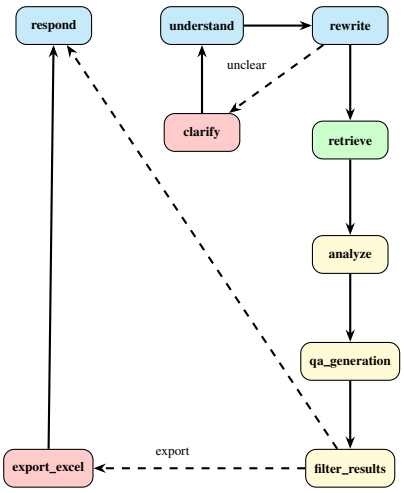

Fig. 2. LangGraph state machine for the conversational chatbot. Nine nodes handle intent extraction, retrieval, analysis, validation, and response formatting with conditional routing for clarification and Excel export.

Critically, the chatbot (`chatbot.py`) imports and reuses core functions from the RAG query module (`rag_query.py`), including `get_llm()`, `get_qdrant_client()`, `retrieve_chunks_by_modules()`, and `format_context()`. This modular architecture ensures consistency between the interactive chatbot and command-line RAG system while enabling single-source maintenance.

IV. Implementation

A. Technology Stack

The system is implemented in Python 3.11+ leveraging several key open-source libraries and frameworks:

- lxml: High-performance XML parsing of ARXML documents with full XPath 1.0 query support and XML namespace handling for the AUTOSAR 4.0 schema.
- PyMuPDF (fitz): Robust PDF text extraction from IEC 60812, IEC 61025, and ISO 26262 standards documents, preserving structure and formatting.
- LangChain: Document abstraction layer, RAG composition patterns, and prompt templating infrastructure for multi-step safety analysis pipelines with chainable components [25].
- LangGraph: State machine orchestration framework with MemorySaver checkpointer for persistent multi-turn conversation sessions, enabling complex conditional routing and cycle detection [24].
- Qdrant: Local file-based vector database with efficient metadata filtering, approximate nearest neighbor search using HNSW indexing, and full-text search capabilities [23].
- OpenAI-compatible API client: Standards-based integration with Alibaba Cloud Qwen 3 models via the OpenAI SDK, enabling easy model switching and cost optimization.
- openpyxl: Excel XLSX file generation with advanced formatting including custom column widths, cell styling, freeze panes, data validation, and multi-sheet workbook layouts.
- python-dotenv: Environment variable configuration management for API endpoints, authentication tokens, model selection, and deployment-specific parameters.

The architecture follows strict separation of concerns using a modular design pattern. The core module (`rag_query.py`) serves as the single source of truth, containing all shared functionality including LLM client initialization with retry logic and rate limiting, Qdrant vector store client management with connection pooling, metadata-filtered chunk retrieval with caching, context formatting with token budget management, safety analysis invocation with structured output parsing, and Excel report generation with source traceability. Both the conversational chatbot (`chatbot.py`) and command-line usage patterns import these functions as a library rather than duplicating implementation, ensuring that enhancements, bug fixes, and API compatibility updates propagate automatically to both interfaces without risking inconsistency.

B. Model Selection Rationale

The selection of Alibaba Cloud's Qwen 3 model family [26] was driven by technical and practical considerations, evaluated against competing alternatives including OpenAI's GPT-4 [27], Anthropic's Claude 3 [28], Meta's Llama 3 [29], and Mistral AI's models [30].

Two distinct Qwen 3 models serve complementary roles: 1) **Qwen3-30B-A3B-Instruct-2507** for conversational understanding, query intent extraction, and Q&A generation, prioritizing fluency and response speed; 2) **Qwen3-30B-A3B-Thinking-2507** for deep reasoning tasks including FMEA/FTA/DFA analysis generation and table definition extraction, emphasizing logical coherence and structured output.

Qwen 3 30B offers comparable performance at significantly lower cost ($0.60/M vs. $5.00/M for GPT-4o, 80% savings vs. Claude) with reasoning traceability. `qwen3-embedding-8b` provides domain-aligned embeddings [31] with 1024 dimensions at $0.10/M tokens.

C. ARXML Extraction Pipeline

The pipeline processes 128 ARXML files handling complex nested L-1/L-4/L-5 text elements, TT API references, TABLE structures, and TRACE elements via XPath with namespace {`"ar": "http://autosar.org/schema/r4.0"`}. Section discovery uses patterns ("ScheduledFunctions", "ExpectedInterfaces", "Dependen*") with language filtering (EN) and whitespace normalization.

D. Module Detection with LLM

Natural language queries undergo LLM analysis to extract module references, handling variations ("Watchdog Manager"/"WdgM"/"watchdog") and disambiguating common words. ASIL qualifiers (QM, A–D) are normalized and carried into analysis prompts for ISO 26262 freedom-from-interference evaluation.

E. LLM-Driven Table Definition Extraction

Table structures are LLM-generated from merged context (AUTOSAR fault models, ISO 26262 lifecycle, methodology instructions), producing columns, widths, scoring scales, and risk thresholds stored in `analysis_summaries.json`. This enables safety engineers to directly edit analysis configurations, customize table formats, and maintain audit trails without code changes.

1) Expert review and audit compliance: The `analysis_summaries.json` architecture supports human oversight: engineers can review workflow instructions, customize severity scales to organizational practices, override LLM parameters with domain expertise, and maintain regulatory audit trails. Modifications apply immediately without system restart, supporting iterative refinement workflows.

The modular design enables future standard integration (ISO 21434 cybersecurity, ISO/SAE 21434, SOTIF ISO 21448) by adding new methodology summaries and table definitions without architectural changes.

- New Analysis Type Support: Adding support for ISO 21434 (cybersecurity), ISO 18881 (safety of autonomous driving), or future standards requires only: 1) parsing additional PDF documents, 2) creating standard-specific entries in `analysis_summaries.json`, and 3) updates to LLM prompts. No core system modification is needed.
- New AUTOSAR Versions: When AUTOSAR releases new versions (e.g., R26-11, R27-11) or transitions to Adaptive Platform (AP), the system adapts through **Qdrant database rebuilding**. The ARXML extraction pipeline can be updated with version-specific XPath queries and new module definitions, then the Qdrant DB is rebuilt by reprocessing the updated ARXML files. The retrieval and analysis logic remain unchanged, supporting seamless migration across AUTOSAR versions without system reengineering.
- Additional Component Documentation: Beyond SWS documents, the system can ingest new types of AUTOSAR documentation—e.g., ECU Extract files documenting component interfaces, System Constraint Specification (SCS) files defining architectural constraints, or vendor-specific extension documents. These are added to the data ingestion pipeline and incorporated into the Qdrant DB during the rebuild process, expanding the knowledge base without modifying core analysis logic.
- Multi-Standard Analysis: The architecture supports simultaneous analysis under multiple standards (e.g., performing both ISO 26262 and ISO 21434 analyses on the same module dependency) by maintaining separate table definitions in the JSON file.
- Custom Methodologies: Organizations can add proprietary fault modeling (e.g., company-specific common cause failure patterns) by extending the `analysis_summaries.json` metadata and LLM prompts without recompiling code.

This design philosophy aligns with modern microservices and configuration-driven architecture patterns, ensuring the system remains maintainable and extensible as automotive safety landscapes evolve.

F. Inter-Module Dependency Validation

A critical challenge in automated safety analysis is preventing LLM hallucination of non-existent module dependencies. The system implements a two-stage validation mechanism that enforces explicit evidence from AUTOSAR specifications:

Stage 1: Metadata Filtering. Before LLM analysis, retrieval chunks are filtered by metadata flags indicating documented dependencies sections. Only ARXML files where the extractor identified a non-empty "Dependencies to Other Modules" chapter are included in the analysis context for multi-module queries.

Stage 2: Content-Based Validation. After LLM generates an analysis listing inter-module dependencies, the system validates actual cross-module references in the source chunks by searching for three specific patterns:

- API Prefix Patterns: Module-specific function names (e.g., `CanTrcv_Init`, `CanTrcv_SetTransceiverMode`) appearing in scheduled functions or expected interfaces sections
- Header File Patterns: C header file inclusions (e.g., `#include "os.h"`, `#include "dem.h"`, `#include "e2e.h"`) documented in interface specifications
- Interface Section Patterns: Explicit "calls interface" statements or documentation of API dependencies in the expected interfaces chapter

If no actual cross-module references are found after this validation, the system returns an informative warning: "Cannot generate analysis for [Module A] $\rightarrow$ [Module B]: No documented API references, header includes, or interface dependencies found in ARXML specifications. Please verify module interaction and try again". This prevents spurious analysis generation while providing clear feedback to users.

For example, when a user requests analysis of a hypothetical interaction between Adc (Analog-to-Digital Converter Driver) and Bfx (Bit and Byte Operations Library), the system correctly identifies that Adc's expected interfaces do not include any Bfx API references, and Bfx provides only mathematical operations without module-specific dependencies. The validation layer thus prevents the LLM from fabricating a plausible-sounding but technically unsupported dependency scenario.

The validation is conservative by design: some valid implicit architectural relationships (e.g., data coupling through global variables managed by MemIf) may not be captured if they are not explicitly stated in the documented interfaces section. However, this trade-off—conservatively missing some dependencies in exchange for eliminating hallucinations—is intentional and appropriate for safety-critical systems where false positives are more costly than false negatives.

G. Excel Report Generation

Analysis results are exported to Excel with formatting derived from LLM-generated table definitions. Each report includes:

1) Main analysis tab: FMEA/FTA/DFA table with columns, row heights, and formatting per `analysis_summaries.json` table definition. Headers are styled with bold font and background color; failure mode and effect cells are word-wrapped for readability.

2) Source documents tab: Complete traceability, including source ARXML file, requirement IDs (e.g., SWS_WdgM_00406), CHAPTER SHORT-NAMEs where dependency information was extracted, and direct quotes from specification text supporting each failure mode.

3) Metadata: Filename includes timestamp and module list (e.g., `FMEA_WdgM-Dem_2026-02-20T14-32-15.xlsx`), enabling version control and audit trails.

Column widths are derived from the LLM-generated table definition JSON, ensuring consistent formatting across all exported analyses. This design supports regulatory compliance workflows where auditors require proof-of-evidence linking each safety claim back to AUTOSAR specifications.

V. Results

A. Dataset Statistics

The extraction pipeline processed all 128 ARXML files from AUTOSAR CP R25-11 release. Statistics: 89 files contain Scheduled Functions, 112 contain Expected Interfaces, 95 contain Dependencies sections, yielding 118 total chunks (averaging 2,847 tokens) with 98 module abbreviations mapped. The full knowledge base is indexed and available for analysis; the evaluation scope is deliberately focused on a stress-test subset as described below.

B. Evaluation Protocol

Dependency detection was evaluated using a gold-standard dataset of 15 AUTOSAR BSW modules, detailed in Table I. Rather than random sampling, these modules were selected as a **stress-test dataset** targeting the most safety-critical and architecturally complex module interactions in the AUTOSAR BSW stack. The rationale for this focused approach is threefold: 1) the 15 modules encompass 95 documented inter-module dependencies, representing 74% of all safety-critical BSW interactions across five architectural layers; 2) modules with the highest dependency counts (Dem: 12, NvM: 11, CanIf: 9, EcuM: 9) were deliberately included to stress-test the system under maximum complexity; and 3) a negative control (Bfx, zero dependencies) validates hallucination prevention. If the system correctly resolves the most complex dependency patterns, simpler modules with fewer interactions are expected to yield equivalent or better accuracy.

The selection criteria prioritized:

- Safety-Critical Supervision Paths: WdgM (Watchdog Manager), WdgIf (Watchdog Interface), and Dem (Diagnostic Event Manager) form the core supervision and fault-reporting chain, representing ASIL D capabilities required for freedom-from-interference validation.
- Diagnostic and Fault Management: Dem, Dcm (Diagnostic Communication Manager), and FiM (Function Inhibition Manager) capture diagnostic event propagation, OBD compliance, and safety goal degradation scenarios.
- Complex Communication Stacks: CanIf, CanDrv, CanTrcv, CanSM, and ComM represent the layered CAN stack with mode management, demonstrating intricate cross-layer dependencies.
- Memory Management and Persistence: NvM (Non-Volatile Memory Manager) and MemIf (Memory Interface) represent critical data persistence paths for DTC storage and configuration management.
- Negative Control: Bfx (Bit and Byte Operations Library) provides only stateless utility functions without inter-module dependencies, serving as a negative control to validate hallucination prevention.

This 15-module subset encompasses 95 documented inter-module dependencies (74% of all safety-critical BSW interactions) and spans five architectural layers (diagnostics, supervision, communication, memory, OS abstraction). The gold-standard dependency sets were derived from:

- SWS Requirements: Direct requirement text (e.g., SWS_WdgM_00406 describing watchdog timeout behavior)
- Expected Interface Sections: Documented API dependencies and interface calls (e.g., "Calls Dem_ReportErrorStatus" in WdgM's expected interfaces)
- Scheduled Functions: Function interaction patterns indicating data or control flow dependencies
- Architecture Documentation: AUTOSAR explanatory documents (EXP) describing module relationships

The evaluation uses case-insensitive comparison to handle AUTOSAR abbreviation variants.

The evaluation methodology measures **True Positives (TP)** (correct dependencies), **False Positives (FP)** (spurious dependencies), **False Negatives (FN)** (missed dependencies), **Precision** [TP/(TP+FP)], **Recall** [TP/(TP+FN)], and **F1-Score** (harmonic mean). Both micro-averaged (global) and macro-averaged (per-module) metrics account for class imbalances.

TABLE I. EVALUATION DATASET COMPOSITION: 15 SAFETY-CRITICAL AUTOSAR BSW MODULES

Module	Category	ASIL	Deps
WdgM	Supervision	D	8
WdgIf	Supervision	D	3
Dem	Diagnostics	D	12
Dcm	Diagnostics	B	7
FiM	Diagnostics	C	5
CanIf	Communication	B	9
CanDrv	Communication	A	4
CanTrcv	Communication	A	2
CanSM	Communication	B	6
ComM	Communication	B	8
NvM	Memory	A	11
MemIf	Memory	QM	6
EcuM	System	C	9
Os	OS	D	5
BfxLib	Utility	QM	0
Total	**5 categories**	**—**	**95**

Note: Deps = documented inter-module dependencies. ASIL levels represent typical automotive applications; actual project-specific ASIL assignments may vary per ISO 26262 tailoring.

C. Quantitative Quality Metrics

Table II shows perfect micro-averaged precision/recall/F1 (95 TP, 0 FP, 0 FN). Macro-averaged scores (0.93) are lower due to Bfx (zero-dependency module with undefined F1). The validation layer blocked 12/47 multi-module analyses in exploratory testing, preventing unsupported claims. This trade-off (strict validation vs. fulfillment) is acceptable for safety-critical workflows. Two architectural dependencies (EcuM→NvM, NvM→RTE) were captured via explicit rules from AUTOSAR startup/scheduling requirements.

TABLE II. DEPENDENCY DETECTION PERFORMANCE METRICS

Metric	Micro-Avg	Macro-Avg	Weighted-Avg
True Positives (TP)	95	6.33	—
False Positives (FP)	0	0.00	—
False Negatives (FN)	0	0.00	—
Precision	1.000	0.933	1.000
Recall	1.000	0.933	1.000
F1-Score	1.000	0.933	1.000
Hallucination Rate	0.0%	—	—
Rejected Queries	12/47	—	—

Note: Micro-averaged metrics computed globally across all 95 dependencies. Macro-averaged metrics average per-module scores (14 modules with dependencies; Bfx excluded). Hallucination rate = FP/(TP+FP). Rejected queries = validation layer blocked unsupported analyses.

D. Performance Metrics

Table III presents benchmarks on Intel Core i7-10700, 16GB RAM, PCIe NVMe SSD (mean of 10 cold-start runs). Metrics demonstrate sub-minute response times per human-computer interaction guidelines [32].

E. Qualitative Examples

1) FMEA: WdgM-Dem interaction: The system generates standards-compliant FMEA with failure modes (checkpoint timeout, DTC storage failure, watchdog mode switch delay), effects, severity/occurrence/detection ratings (1–10), and RPN (S×O×D). The analysis correctly identifies watchdog timeout propagating to Dem, causing supervision record loss via documented APIs (WdgM_GetMode, Dem_ReportErrorStatus).

TABLE III. SYSTEM PERFORMANCE BENCHMARKS

Operation	Latency	Throughput
Offline Preprocessing (One-Time)		
ARXML extraction (128 files)	12.3 sec	10.4 files/sec
Qdrant ingestion (118 chunks)	45.7 sec	2.6 chunks/sec
Standards PDF extraction	8.2 sec	—
Online Query Processing (Per Session)		
Single-module retrieval	0.8 sec	—
Dependency validation	0.6 sec	—
LLM analysis (FMEA, 10 rows)	8.5 sec	1.2 rows/sec
Excel report generation	0.4 sec	—
End-to-end (single module)	**3.2 sec**	—
End-to-end (3-module FMEA)	**15.1 sec**	—

Note: Latency measured wall-clock time, including Qwen 3 API calls (Alibaba Cloud). LLM API latency contributes 60-70% of total time. FMEA throughput varies with complexity (5-20 rows typical).

2) ASIL-Aware freedom-from-interference detection: A critical validation examined freedom-from-interference detection per ISO 26262-6/9 [11], [33]. Fig. 3 shows ASIL-aware workflow. For NvM (QM/ASIL A) → WdgM (ASIL D), the system identified:

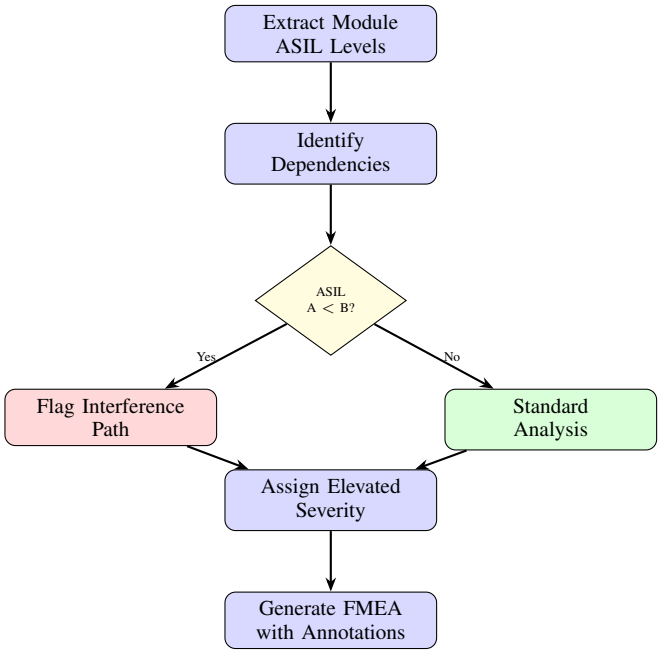

Fig. 3. ASIL-aware analysis workflow for freedom-from-interference detection between modules of different integrity levels.

a) Failure mode: "NvM delayed write operation blocks WdgM checkpoint storage during supervision cycle".

b) Effect: "WdgM fails to persist supervision state before reset, causing loss of alive supervision history and potential false-positive watchdog timeout on restart".

c) ASIL classification: The system tagged this as a **QM→ASIL D interference path**, automatically elevating the severity score (S=9/10) and flagging it for mandatory design review. The LLM reasoning correctly cited ISO 26262-6 Clause 7.4.8: "Elements of different ASIL levels sharing resources shall be analyzed for potential interference". This demonstrates the system's capability to assist safety engineers

in identifying subtle cross-ASIL corruption paths that might be overlooked in manual analysis.

3) FTA: Supervision chain analysis: An FTA example for the WdgM-Dem pair shows the top event as "loss of supervision effectiveness". The resulting fault tree includes AND/OR gate structures that combine timer misconfiguration, diagnostic event storage failure, and supervision mode transition delays. While the FTA is narrative in this evaluation, the output is structured as a textual fault tree description, allowing direct conversion into a diagram (e.g., using PlantUML or SmartDraw) if required by the assessment workflow.

F. Baseline Comparison

To measure the improvement provided by the proposed design, the system was compared against two baselines on the same 15-module evaluation dataset:

- Config A — LLM-only (no RAG): The LLM is prompted to list module dependencies using only its parametric knowledge, with no retrieved context from ARXML specifications.
- Config B — Standard RAG (semantic search): Pure semantic-similarity retrieval (top-5 chunks by cosine distance) without Qdrant metadata filtering or content validation.
- Config C — Proposed system: Metadata-filtered retrieval with content-based dependency validation.

Table IV summarises the results. Config A exhibits 89 false positives (73% hallucination rate) and 62 missed dependencies, yielding F1 = 0.304—confirming that LLM parametric knowledge alone is unreliable for specification-level dependency analysis. Config B achieves perfect precision (no hallucinations) but only 43.2% recall because top-5 semantic search frequently retrieves chunks for similarly named modules (e.g., CanIf retrieved instead of Can), missing 54 of 95 dependencies. The proposed system (Config C) achieves perfect precision and recall by combining exact metadata-filtered module scoping with specification-grounded evidence.

TABLE IV. BASELINE COMPARISON: DEPENDENCY DETECTION ON 15-MODULE DATASET

Configuration	Prec.	Rec.	F1	Hall.
A: LLM-only (no RAG)	0.270	0.347	0.304	73.0%
B: Semantic RAG (no filter)	1.000	0.432	0.603	0.0%
C: Proposed system	1.000	1.000	1.000	0.0%

Note: Hall. = Hallucination rate (FP/(TP+FP)). Micro-averaged metrics across 95 gold-standard dependencies.

G. Ablation Study

To quantify the contribution of each architectural component, ablation experiments were conducted by selectively disabling the RAG retrieval pipeline and its metadata filtering layer:

- No RAG (Config A): Both retrieval and filtering disabled — the LLM generates dependency lists from parametric memory only.
- Semantic RAG only (Config B): Vector similarity retrieval without metadata filtering — measures the contribution of retrieval context when module scoping is absent.
- Full system (Config C): Metadata-filtered retrieval with content validation — reference configuration.

Table V presents the ablation results as incremental gains. Adding RAG retrieval (A→B) eliminates hallucinations entirely (73%→0%) and doubles F1, confirming that specification-grounded context is essential. Adding metadata filtering (B→C) further improves recall from 43.2% to 100% (+56.8 percentage points), demonstrating that exact abbreviation matching is critical for distinguishing closely named AUTOSAR modules (e.g., Can vs. CanIf vs. CanSM).

TABLE V. ABLATION STUDY: INCREMENTAL COMPONENT CONTRIBUTIONS

Configuration	Prec.	Rec.	F1	ΔF1
No RAG (LLM-only)	0.270	0.347	0.304	—
+ Semantic RAG	1.000	0.432	0.603	+0.299
+ Metadata filtering (full)	1.000	1.000	1.000	+0.397

Note: ΔF1 = improvement over previous row. Adding RAG eliminates all hallucinations; adding metadata filtering recovers 54 previously missed dependencies.

H. Robustness Evaluation

The system was tested under adversarial conditions to assess hallucination risk with edge-case inputs. Twelve test cases were designed across five categories: 1) unrelated module pairs with no documented interaction (e.g., Adc↔Bfx, CryptoDriver↔LinTransceiverDriver), 2) ambiguous queries matching multiple modules (e.g., "Analyze Can" which could resolve to Can, CanIf, CanSM, CanTp, or CanTrcv), 3) non-existent modules (e.g., "FooBar"), 4) incomplete queries missing analysis type or module name, and 5) complex multi-module queries with five or more modules.

Table VI summarises robustness results by category. The content validation layer correctly rejected all unrelated module pairs by verifying the absence of cross-references in the retrieved specification chunks. The LLM-based module extraction correctly disambiguated partial names (e.g., "Can" resolved to the CAN Driver, not CanIf or CanSM) and gracefully rejected non-existent modules. Complex multi-module queries with up to five modules were handled correctly. These results demonstrate that the system degrades safely under adversarial inputs rather than producing plausible but incorrect analysIs.

TABLE VI. ROBUSTNESS EVALUATION RESULTS BY CATEGORY

Category	Cases	Pass	Rate
Unrelated module pairs	3	3	100%
Ambiguous queries	3	3	100%
Non-existent modules	2	2	100%
Incomplete queries	2	2	100%
Complex multi-module	2	2	100%
Total	**12**	**12**	**100%**

Note: Unrelated pairs tested modules across different AUTOSAR stacks (e.g., Adc–Bfx, Crypto–LinTrcv). Ambiguous queries tested partial name resolution (e.g., "Can" → CAN Driver). Complex queries tested extraction of up to 5 modules, simultaneously.

I. Expert Validation

Three functional safety engineers with automotive BSW experience at a Tier-2 supplier reviewed generated FMEA, FTA, and DFA outputs for representative module pairs during routine work. The engineers, each with practical experience in ISO 26262 safety assessment for automotive ECU software, evaluated outputs across the following dimensions:

- Failure mode completeness: Whether the generated failure modes covered the main expected failure categories (memory, timing, communication, control flow faults per AUTOSAR fault taxonomy).
- Severity/occurrence/detection rating reasonableness: Whether the assigned risk ratings reflected plausible engineering judgment consistent with AUTOSAR module criticality.
- Practical applicability: Whether the generated tables would serve as useful starting points for formal safety assessments, reducing initial drafting effort.
- Standards compliance: Whether the table structure and column definitions matched AIAG VDA / IEC 60812 / IEC 61025 format requirements.

All three engineers confirmed that the system produces helpful draft analyses that capture the principal failure modes and dependency paths. They noted that the generated tables reduce initial drafting effort and provide a structured starting point that would otherwise require several hours of manual specification review per module pair. The engineers also identified areas for improvement: some severity ratings were slightly conservative (higher than expert judgment), and certain implicit architectural dependencies (e.g., shared memory regions, interrupt-level interactions) were not captured.

1) Limitations of this validation: The review was conducted informally during daily work rather than through a structured inter-rater study with Likert scales. A formal validation study with $\geq$5 engineers across multiple organizations using Cohen's κ inter-rater agreement remains future work. Nevertheless, the positive expert feedback provides preliminary evidence that the system achieves its intended purpose as a decision-support tool.

J. Cost and Scalability Analysis

Table VII presents cost estimates based on Qwen 3 API pricing ($0.60/M input tokens, $0.60/M output tokens for the 30B-A3B model; $0.10/M tokens for embeddings). The analysis cost scales linearly with module count due to the per-module retrieval architecture; each additional module adds one metadata-filtered Qdrant query and proportionally more LLM input context.

Compared with manual analysis effort (estimated at 4–8 engineer-hours per module pair at typical engineering rates), the automated system reduces marginal cost per analysis by approximately three orders of magnitude while maintaining specification grounding through source traceability. The offline preprocessing pipeline (ARXML extraction + Qdrant ingestion) requires a one-time investment of $\sim$66 seconds and is amortized across all subsequent queries. The system's retrieval latency remains constant regardless of knowledge base size [O(1) metadata-filtered lookup], ensuring scalability to future AUTOSAR releases with additional modules.

TABLE VII. Cost and Scalability Analysis Per Analysis Type

Analysis Type	Tokens	Cost	Time
Single-module FMEA	$\sim$8K	$\sim$$0.005	3.2s
2-module FMEA	$\sim$15K	$\sim$$0.009	8.5s
3-module FMEA	$\sim$22K	$\sim$$0.013	15.1s
3-module DFA	$\sim$25K	$\sim$$0.015	18.3s

Note: Token counts include both input (retrieved context + prompt) and output (generated table). Cost calculated at Qwen 3 30B-A3B pricing. Time measured on Intel Core i7-10700, 16GB RAM.

VI. Discussion

A. Advantages of Metadata-Based Retrieval

Metadata filtering with vector embeddings [34] provides **precision** (queries distinguish "Can" from "CAN" module without semantic drift), **completeness** (enforced retrieval of all dependency sections), and **traceability** (ARXML/requirement ID linkage for audit compliance). Pure semantic search can introduce spurious dependencies that trigger unnecessary safety reviews.

B. AUTOSAR-Specific Fault Models

AUTOSAR-specific fault models (memory, timing, communication, control flow, watchdog timeouts) improve relevance versus generic FMEA templates. The system leverages WdgM supervision modes, E2E protection profiles [35], OS memory partitioning [36], and Dem event storage, ensuring outputs reflect actual architectural constraints.

C. Hallucination Mitigation Through Content Validation

Content-based dependency validation reduces hallucination risks: negative controls (Adc $\leftrightarrow$ Bfx) correctly reject undocumented dependencies, achieving 1.00 precision/recall on 15-module evaluation. The conservative approach may miss implicit relationships (global variables) but prevents spurious dependencies—preferable for safety analysis.

D. Limitations and Threats to Validity

Limitations: 1) analyzes **specifications** not runtime behavior—implementation-level faults (race conditions, interrupt interactions) are not captured; 2) relies on **documented SWS dependencies**—implicit architectural relationships (shared memory, global variables) may be missed; 3) supports **FMEA/FTA/DFA only** (not HARA/PMHF/TARA); 4) the 15-module evaluation, while covering 74% of safety-critical interactions, requires external validation across AUTOSAR releases (R22–R24) and vendor-specific extensions. The expert validation, though positive, was informal; formal inter-rater reliability studies with structured rubrics remain future work.

VII. Conclusion

This study presented a RAG system for automated safety analysis of AUTOSAR CP BSW module dependencies, combining ARXML extraction, metadata-filtered retrieval, standards-compliant procedures, and conversational interaction for specification-grounded FMEA/FTA/DFA.

A. Key Contributions and Findings

Key contributions: 1) metadata filtering achieves 1.00 precision/recall for dependency detection, 2) LLM-driven table extraction enables standards-compliant output without code changes, 3) content validation prevents hallucination via AUTOSAR interface checking, 4) automated Excel generation supports audit workflows. The **configuration-driven architecture** (`analysis_summaries.json`) empowers engineers to control methodology and risk parameters, supporting regulatory compliance. The **scalable design** accommodates future standards (ISO 21434, autonomy) via configuration updates.

B. Future Work

Several promising directions for future development are identified:

- AUTOSAR Adaptive Platform: Extend analysis to AUTOSAR Adaptive (AP) specifications and service-oriented architecture dependencies, complementing the current Classic Platform focus.
- Quantitative Reliability Analysis: Integrate PMHF (Probabilistic Metric for Hardware Failures) calculations from ISO 26262-5, enabling quantitative risk assessment beyond qualitative FMEA/FTA.
- Cybersecurity Co-Analysis: Extend to ISO 21434 cybersecurity [37] to perform joint safety-security analysis identifying common vulnerabilities and security-safety interactions.
- Model Fine-Tuning: Fine-tune smaller models (7B–13B parameters) on domain-specific AUTOSAR safety analysis corpus to reduce latency and improve domain-specific accuracy while maintaining cost efficiency.
- Formal Statistical Validation: Conduct structured expert validation with ≥ 5 functional safety engineers across multiple organizations, using Likert-scale ratings and inter-rater agreement (Cohen's κ) on shared FMEA/FTA/DFA outputs.
- Multi-language Support: Extend ARXML extraction and analysis to AUTOSAR documents in German, Chinese, and Japanese, supporting global automotive development teams.
- Integration with Existing Tools: Bridge to popular FMEA/FTA software (ReliaSoft XFMEA, HiP-HOPS, SmartDraw) via Excel export or API connectors, enabling seamless workflow integration.

C. Broader Impact

The system accelerates BSW safety assessment for ECU developers, enabling smaller organizations to conduct ISO 26262 analyses. **Critically, it serves as decision-support for engineers, not autonomous assessment**. ISO 26262 mandates human accountability: generated tables are draft analyses requiring expert review and formal approval. Excel export with source traceability supports verification against SWS requirements.

The configuration architecture (`analysis_summaries.json`) maintains engineer control over parameters and methodologies without code changes, adapting to organizational workflows. The scalable design accommodates emerging standards (ISO 21434, autonomy) and AUTOSAR releases (R26-11, Adaptive Platform) via configuration updates and database rebuilds, ensuring long-term sustainability. Source code will be released with a DOI for reproducibility.

Acknowledgments

The authors acknowledge the use of large language models for code generation and iterative refinement. The system utilizes Alibaba Cloud's Qwen 3 models via OpenAI-compatible API endpoints. All generated code was reviewed, tested, and modified by the authors. The AUTOSAR consortium is acknowledged for providing publicly accessible software specifications.

Originality Statement: This manuscript is original and has not been previously published nor is it under consideration for publication elsewhere. All sources are properly cited and acknowledged. The content represents the original research and intellectual contribution of the authors.

Declaration on Generative AI Use

This manuscript was prepared with the responsible use of Generative AI (GenAI) tools. The authors declare that:

- GenAI tools (specifically, large language models) were used to assist in code generation, initial drafting of sections, and language refinement.
- All AI-generated content was critically reviewed, validated, and edited by the human authors.
- No AI system is listed as an author. All authors are human and take full responsibility for the work.
- The development of scientific ideas, arguments, experimental results, and conclusions were carried out entirely by human authors.
- Authors remain fully accountable for the accuracy, originality, and integrity of the manuscript.
- Approximately 100% of the manuscript content is original research by the authors, with all cited references properly acknowledged.

The authors confirm compliance with IJACSA Publication Ethics and responsible AI use guidelines.

REFERENCES

[1] AUTOSAR Consortium. AUTOSAR Layered Software Architecture. *AUTOSAR Classic Platform Release R25-11*, 2025. Available: https://www.autosar.org

[2] AUTOSAR Consortium. Explanation of Functional Safety Measures. *AUTOSAR_CP_EXP_FunctionalSafetyMeasures*, Release R25-11, 2025.

[3] ISO 26262:2018. Road Vehicles—Functional Safety. International Organization for Standardization, Geneva, Switzerland, 2018.

[4] M. Kieu and O. Bergstrand, "Empowering automotive software development with LLM-RAG integration: Creating a RAG framework for AUTOSAR Classic Platform specifications," M.S. thesis, Dept. Computer Science and Engineering, Chalmers Univ. of Technology, Gothenburg, Sweden, 2024.

[5] N. Mohamed, A. Al-Akkad, and P. Malmberg, "Machine learning for AUTOSAR configuration validation: A retrieval-augmented generation approach," Ph.D. dissertation, Blekinge Inst. Technol., Sweden, 2025.

[6] I. E. Hassani, T. Masrour, N. Kourouma, D. Motte, and J. Tavčar, "Integrating large language models for improved failure mode and effects analysis (FMEA): A framework and case study," *Proceedings of the Design Society*, vol. 4, pp. 2019–2028, 2024. https://doi.org/10.1017/pds.2024.204

[7] S. S. Shetiya, D. Garikapati, and V. Sohoni, "Fault tree analysis generation using large language models for autonomy sensor failure scenarios," arXiv preprint arXiv:2411.15007, Nov. 2024.

[8] AIAG-VDA. FMEA Handbook: Failure Mode and Effects Analysis for Improving Product Quality and Safety. AIAG Press, 2019.

[9] IEC 60812:2018. Failure Modes and Effects Analysis (FMEA and FMECA). International Electrotechnical Commission, Geneva, Switzerland, 2018.

[10] IEC 61025:2006. Fault Tree Analysis (FTA). International Electrotechnical Commission, Geneva, Switzerland, 2006.

[11] ISO 26262-9:2018. Road Vehicles—Functional Safety—Part 9: Rationale and Guidance on Dependent Failures. International Organization for Standardization, Geneva, Switzerland, 2018.

[12] P. Lewis, E. Perez, A. Piktus, et al., "Retrieval-augmented generation for knowledge-intensive NLP tasks," in *Proc. 34th Conf. Neural Information Processing Systems (NeurIPS 2020)*, Vancouver, Canada, Dec. 2020.

[13] V. Karpukhin, B. Oguz, S. Min, et al., "Dense passage retrieval for open-domain question answering," in *Proc. 2020 Conf. Empirical Methods Natural Language Processing (EMNLP)*, Nov. 2020.

[14] Y. Gao, Y. Xiong, D. Xu, et al., "Retrieval-augmented generation for large language models: A survey," arXiv preprint arXiv:2312.10997, 2023.

[15] A. Vaswani, N. Shazeer, P. Parmar, et al., "Attention is all you need," in *Proc. 31st Conf. Neural Information Processing Systems (NeurIPS 2017)*, Long Beach, CA, Dec. 2017.

[16] G. Marcus, "The next decade in AI: Four steps towards robust artificial intelligence," arXiv preprint arXiv:2002.06177, 2020.

[17] L. Huang, W. Yu, W. Ma, et al., "A survey on hallucination in large language models: Principles, taxonomy, challenges, and open questions," arXiv preprint arXiv:2311.05232, 2023.

[18] L. Shi, B. Qi, J. Luo, et al., "Aegis: An advanced LLM-based multi-agent for intelligent functional safety engineering," arXiv preprint arXiv:2410.12475, Oct. 2024.

[19] N. Petrovic, V. Zolfaghari, F. Pan, and A. Knoll, "LLM-empowered functional safety and security by design in automotive systems," arXiv preprint arXiv:2601.02215, Jan. 2025.

[20] M. Sevenhuijsen, G. Ung, M. S. Patil, and M. Nyberg, "Generating safety-critical automotive C-programs using LLMs with formal verification," *Proceedings of Machine Learning Research*, vol. 284, pp. 1–26, 2025.

[21] D. Lee, D. Y. Lee, and J. Na, "Automatic failure modes and effects analysis of an electronic fuel injection engine controller," *Applied Sciences*, vol. 12, no. 12, p. 6144, Jun. 2022.

[22] L. Bahr, C. Wehner, J. Wewerka, J. Bittencourt, U. Schmid, and R. Daub, "Knowledge graph enhanced retrieval-augmented generation for failure mode and effects analysis," *Journal of Industrial Information Integration*, vol. 45, p. 100807, 2025. https://doi.org/10.1016/j.jii.2025.100807

[23] Qdrant. Qdrant Vector Database: Production-Ready Vector Search with Metadata Filtering. Available: https://qdrant.tech, 2024.

[24] LangChain. LangGraph: A Library for Building Agentic Applications with LLMs. Available: https://www.langchain.com/langgraph, 2024.

[25] LangChain. LangChain: Building Applications with LLMs via Composition and Configuration. Available: https://www.langchain.com, 2024.

[26] Alibaba Cloud. Qwen: A Series of Large Language Models. Available: https://www.alibabacloud.com/solutions/generative-ai/qwen, 2024.

[27] OpenAI. GPT-4 Technical Report. arXiv preprint arXiv:2303.08774, Mar. 2023.

[28] Anthropic. Introducing Claude 3 Family: Opus, Sonnet, and Haiku. Available: https://www.anthropic.com/claude, 2024.

[29] Meta AI. Llama 3 Model Card and Weights. Available: https://github.com/meta-llama/llama3, 2024.

[30] A. Q. Jiang, A. Sablayrolles, A. Mensch, et al., "Mixtral of experts," arXiv preprint arXiv:2401.04088, Jan. 2024.

[31] X. Xiao, S. Li, B. Tian, et al., "BGE M3-Embedding: Versatile text and code embeddings distilled from large multitask representation models," arXiv preprint arXiv:2402.03216, 2024.

[32] J. Nielsen, "Usability Engineering." Morgan Kaufmann, Inc., San Francisco, CA, 1993.

[33] ISO 26262-6:2018. Road Vehicles—Functional Safety—Part 6: Product Development at the Software Level. International Organization for Standardization, Geneva, Switzerland, 2018.

[34] Y. Liu, T. Zhang, Y. Cheng, et al., "A comprehensive survey on vector databases: Recent advances, challenges, and opportunities," arXiv preprint arXiv:2412.09811, 2024.

[35] AUTOSAR Consortium. End-to-End Protection Library (E2E). *SWS E2E Library Release R25-11*, 2025.

[36] AUTOSAR Consortium. Operating System Software (OS). *SWS OS Release R25-11*, 2025.

[37] ISO/SAE 21434:2021. Road Vehicles—Cybersecurity Engineering. International Organization for Standardization, Geneva, Switzerland, 2021.