10 APPENDIX A - SUMMARY TABLES

Paper	Tool / Framework / Technique	Description
[18]	LLM-based Code Verification	Uses LLMs like GPT-3.5 to verify code by analyzing requirements and explaining whether they are met.
[17]	nl2spec	A framework leveraging LLMs to generate formal specifications from natural language, addressing ambiguity in system requirements with iterative refinement.
[73]	Explanation-Refiner	A neuro-symbolic framework integrating LLMs and theorem provers to formalize and validate explanatory sentences, providing error correction and feedback for improving NLI models.
[50]	*Not Specified*	Analyzes research directions in software requirement engineering, conducting a SWOT analysis and sharing evaluation findings.
[5]	Symbolic NLP vs. ChatGPT	Compares the performance of symbolic NLP and ChatGPT in generating correct JML output from natural language preconditions.
[4]	Domain Model Extractor	Generates domain models from natural language requirements in an industrial case study, evaluating accuracy and performance.
[59]	SpecSyn	A framework using LLMs for automatic synthesis of software specifications, improving accuracy by 21% over previous tools.
[23]	AssertLLM	A tool generating assertions for hardware verification from design specifications using three customized LLMs, achieving 89% correctness.
[30]	Formal Verification of NASA's Software	Reports on formal verification of NASA's Node Control Software natural language specifications, highlighting errors and lessons learned.
[54]	SpecLLM	Explores using LLMs for generating and reviewing VLSI design specifications, improving chip design documentation.
[68]	Requirements Specification Language (RSL), ReDSeeDS	Enhanced software requirements specification using constrained natural language and automated transformations into code.
[31]	ARSENAL Framework and Methodology	Automated extraction of requirements specification from natural language with automatic verification.
[51]	BPM-to-NL Translation Process	Generated natural language descriptions from business process models for better validation.
[61]	Laurel	A framework to generate Dafny assertions to automate program verification process for a SMT solver

Table 1. Summary of LLMs related literature: Tools, Frameworks, and Achievements

Paper	Tool / Framework / Technique	Description
[61]	Laurel	A framework to generate Dafny assertions to automate program verification process for a SMT solver
[69]	Controlled Natural Language (CL) with ANLT	Expressed software requirements in a limited set of natural language and translated to logical expressions to detect ambiguities.
[74]	LLM-based Analysis for Smart Grid Requirements	Improved smart grid requirement specifications with GPT-4o and Claude 3.5 Sonnet, achieving F1-scores between 79% - 94%.
[91]	NL-to-LTL Translation via LLMs	Converted unstructured natural language requirements to NL-LTL pairs, achieving 94.4% accuracy on public datasets.
[81]	ESBMC-AI	Combined LLMs with Formal Verification to detect and fix software vulnerabilities with high accuracy.
[38]	LLM-based Formal Specifications Translation	Translated natural language into formal rules (regex, FOL, LTL) with high adaptability and performance.
[62]	SynVer Framework	Synthesized and verified C programs using the Verified Software Toolchain.
[72]	LLM-based Requirement Coverage Analysis	Ensured low-level software requirements met high-level requirements, achieving 99.7% recall in spotting missing coverage.
[24]	SAT-LLM	Integrated SMT solvers with LLMs to improve conflict identification in requirements; significantly outperformed standalone LLMs in detecting complex conflicts.
[21]	NLP for Software Development	Assessed NLP techniques for various software development stages, highlighting their suitability for generating assertions and processing developer queries.
[63]	Req2Spec	NLP-based tool that formalises natural language requirements for HANFOR; achieved 71% accuracy in formalising 222 automotive requirements at BOSCH.
[36]	RML (Requirements Modeling Language)	Introduced a conceptual model-based framework ensuring precision, consistency, and clarity in requirements writing.
[22]	GPT-4o for VeriFast Verification	Evaluated GPT-4o's ability to generate C program specifications for VeriFast; found that while functional behavior was preserved, verification often failed or contained redundancies.
[66]	NL to Temporal Logic Translation	Developed an automatic translation mechanism from natural language sentences to temporal logic for formal verification.

Table 2. Summary of LLMs related literature: Tools, Frameworks, and Achievements

Paper	Tool / Framework / Technique	Description
[14]	ANTONIO toolkit	A comprehensive analysis of NLP verification approaches and introduces a structured **NLP Verification Pipeline** with six key components. The work includes identifying gaps in existing methods, proposing novel solutions for improved robustness, extending standard verifiability metrics, and emphasizing the importance of reporting verified subspace (geometric and semantic) properties for better reliability and interpretability.
[89]	Lemur	Integrated LLMs with automated reasoners for program verification, defining sound transition rules and demonstrating improved performance on benchmark tests.
[65]	Systematic Review	Conducted a comprehensive review on natural language to formal specification translation, analyzing research across multiple academic databases.
[67]	LLM-based Safety Requirements Pipeline	Designed a pipeline using LLMs to refine and decompose safety requirements for autonomous vehicles, evaluated through expert assessments and industrial implementation.
[92]	Specification Consistency Framework	Ensured consistency between oral and formal specifications, incorporating time extraction, input-output partitioning, and semantic reasoning, with positive evaluation results.
[64]	NLP Tools for TFM	Evaluated six NLP pipelines for Topological Functioning Modelling (TFM). Found that Stanford CoreNLP, FreeLing, and NLTK performed best.
[60]	LLM-based Dafny Task Generation	Used LLMs (GPT-4, PaLM-2) to generate Dafny tasks from MBPP benchmark using different prompting strategies (context-less, signature, retrieval-augmented CoT). GPT-4 achieved best results with retrieval-augmented CoT prompt, producing 153 verified Dafny solutions.
[93]	LeanDojo & ReProver	Introduced LeanDojo, an open-source toolkit for interacting with the Lean theorem prover. Developed ReProver, a retrieval-augmented LLM-based prover that improved theorem proving efficiency. Created a benchmark with 98,734 theorems and proofs for testing generalization.
[47]	Thor and class methods named Hammers	Introduced a framework named Thor which integrates language models with theorem provers. Hammers are implemented to find the appropriate premises to complete the proofs of conjectures. Datasets used are PISA and MiniF2F.
[35]	Not specified	The work proposes the integration of major formal languages (Dafny, Ada/SPARK, Frama-C, and KeY), their interactive theorem provers (Coq, Isabelle/HOL, Lean) with Copilot.
[56]	Systematic Review	Conducted a comprehensive survey on formal specification and verification of autonomous robotic systems in 2018. This is based on literature available of ten years (2008 - 2018).
[34]	Symbolic analysis and LLMs prompts	The quality of annotations produced in ACSL format is measured for PathCrawler and EVA (tools available in Frama-C). PathCrawler generated more context-aware annotations while, EVA efficiency improved having less run-time errors.

Table 3. Summary of LLMs related literature: Tools, Frameworks, and Achievements

Paper	Tool / Framework / Methodology Devised	Description
[76]	Dynamic Requirements Traceability Model	Proposed a model to improve software quality through verification and validation of functional requirements, addressing scalability for both small and large projects.
[77]	Early Phase Traceability and Verification Model	Introduced a model for early development phase traceability and verification, with enhanced adaptability to requirement changes and impact analysis.
[29]	Comprehensive Review	Reviewed software requirements traceability, covering elements, challenges, techniques, classified approaches, and identified future research directions.
[75]	Empirical Analysis with Traceability Metrics	Conducted an empirical study enforcing requirements completeness, introducing traceability metrics and regression analysis to quantify software quality and reduce defect rates.
[40]	RETRO	Developed RETRO, a tool for automating RTM generation, significantly improving accuracy and efficiency over manual tracing methods.
[84]	VSM + BTM-GA Hybrid Approach	Proposed a hybrid method for traceability link generation that outperformed traditional IR techniques, particularly in agile development, enhancing recall and precision.
[2]	Trustrace	Introduced a trust-based traceability recovery approach using mined repository data, achieving better precision and recall than standard IR methods.
[37]	Deep Learning-Based Traceability (BI-GRU)	Applied deep learning with BI-GRU for traceability, incorporating semantic understanding and domain knowledge, outperforming VSM and LSI methods.
[6]	Topology-Based Model-Driven Approach	Presented a topology-based, model-driven traceability technique formalising specifications and establishing trace links between real-world functions and software artifacts.
[16]	Event-Based Traceability Mechanism	Proposed a mechanism to support software evolution through event-based artifact linking, improving change management performance and maintaining consistency in distributed environments.
[19]	Tracebok	Introduced a traceability body of knowledge framework categorising traceability approaches and offering practical guidance for software projects.
[58]	Traceability Visualisation Review	Reviewed visualisation tools and techniques, identifying issues such as scalability and visual clutter, and suggesting ways to improve traceability visualisation.

Table 4. Summary of Other Sections of Literature Review: Tools, Frameworks, and Methodologies

Paper	Tool / Framework / Methodology Devised	Description
[48]	Zero-Shot CoT	Demonstrated that zero-shot prompts with simple additions like "Let's think step by step" can significantly enhance LLM reasoning without training examples.
[55]	One-Shot Prompting	Used a single example to guide LLMs in generating desired outputs, showing potential as an alternative to zero- and few-shot approaches.
[95]	Few-Shot Evaluation	Found that zero-shot prompting can outperform few-shot setups, challenging assumptions on example-based prompting.
[96]	Self-Consistent Few-Shot Prompting	Showed that prompting with few examples isn't always better and proposed techniques to refine few-shot reliability.
[42]	Prompt Engineering Review	Surveyed prompt engineering strategies, including multimodal prompts, adversarial prompting, and robustness evaluations.
[85]	Chain of Thought (CoT)	Proposed stepwise reasoning in prompts to improve LLM performance in tasks requiring logic, math, and symbolic manipulation.
[42]	Lost-in-the-Middle	Highlighted LLMs' U-shaped attention patterns, warning against long prompts where important middle-context information may be overlooked.
[52]	Retrieval Augmented Generation (RAG)	Introduced a method to retrieve relevant knowledge for augmenting prompts, enhancing performance on knowledge-intensive tasks.
[43]	LoRA	Proposed Low-Rank Adaptation for fine-tuning LLMs efficiently without retraining the entire model, supporting modular adaptability.
[83]	Iterative Prompting Framework	Proposed a context-aware iterative prompting method for LLMs in multi-step reasoning tasks, dynamically synthesizing prompts to improve reasoning accuracy.
[88]	UTP Tutorial	Provided an introduction to Unifying Theories of Programming (UTP) and the use of alphabetised relational calculus to describe imperative constructs such as Hoare logic and refinement calculus.
[32]	Institutions	Introduced institutions as a formal framework for modeling logical systems, including results on signature gluing and constraints for abstract data types, contributing to the theory of specification languages.
[87]	Circus	Described Circus, a language combining CSP, Z, and imperative programming, formalised using UTP to refine concurrent systems.
[39]	Runtime Verification for UAS	Applied runtime verification in UAS traffic management, using formal requirements to validate system safety across subsystems, confirmed via flight simulations.
[12]	UTP-Based RTEMS Verification	Verified RTEMS real-time OS using Promela and SPIN, linking UTP semantics for test generation and discussing future directions in space-grade verification.

Table 5. Summary of Other Sections of Literature Review: Tools, Frameworks, and Methodologies

Paper	Tool / Framework / Methodology Devised	Description
[26]	Isabelle/UTP	Introduced Isabelle/UTP to mechanise UTP semantics, providing formal proof tools for various paradigms, supporting development of automated verification tools.
[46]	UML Testing Profile (UTP)	Discussed use of UML and UTP for model-based testing of embedded systems, presenting algorithms for test artefact generation under resource constraints.
[27]	Java Model of RTEMS	Presented a verified Java model for RTEMS's priority inheritance protocol using Java Pathfinder, fixing known issues like data races and deadlocks.

Table 6. Summary of Other Sections of Literature Review: Tools, Frameworks, and Methodologies