Detecting and Repairing Incomplete Software Requirements with Multi-LLM Ensembles

Mohamad Kassab
Boston University
Boston, U.S.A.
mkassab@bu.edu

Marwan AbdElhameed
New York University Abu Dhabi
Abu Dhabi, U.A.E.
marwan@nyu.edu

***Abstract*—Ensuring complete software requirements specifications (SRS) is critical to preventing costly downstream errors. We present a tool that ensembles three complementary LLMs—DeepSeek Chat, GPT-4o Mini, and Claude Sonnet 4—to detect and suggest remedies for missing requirements. The tool generates a structured domain model and applies parallel external and internal completeness checks through tailored prompts. Users can select LLMs and aggregation methods (majority, weighted, or meta-fusion). Unlike prior single-model approaches, we systematically evaluate aggregation strategies across four diverse SRS domains. In experiments with seeded omissions, single models achieved only 0–52% recall, whereas our ensemble consistently exceeded 75%—reaching up to 100%—with 95–100% plausibility. These results demonstrate the feasibility of multi-LLM ensembles as practical aids—complementing rather than replacing human analysts—and supporting interactive refinement workflows.**

***Index Terms*—Requirements Completeness, Requirements Engineering, Quality Metrics, Large Language Models (LLMs).**

I. Introduction

Incomplete or vague requirements drive roughly 78% of software project failures [12], with gaps in SRSs causing costly downstream errors and rework [6]. Although ISO/IEC 29148:2018 defines completeness as capturing all inputs, outputs, behaviors, and constraints without implicit assumptions, achieving this in practice is notoriously difficult [14]. Natural language dominates requirement specifications (rising from 61% in 2013 to 69% in 2020 [8]), making omission detection via manual checklists or reviews labor-intensive and unscalable [11].

A recent systematic literature review we conducted identified six key research questions on requirements completeness—spanning definitions, detection, metrics, improvement, tool support, and the role of AI—and underscored the transformative potential of LLMs for automating completeness checks [7]. Building on these insights, we introduce a tool that ensembles three complementary LLMs: DeepSeek Chat, GPT-4o Mini, and Claude Sonnet 4. These models were selected for their diverse analysis strengths: structured extraction, logical consistency checks, and contextual reasoning. By configurable aggregation and automated domain modeling generation (classes, attributes, relationships), the tool aims to recover missing requirements at scale while preserving precision. To our knowledge, this is the first empirical study that benchmarks multi-LLM ensembles for completeness detection across multiple domains.

We describe the tool's architecture in Section II and evaluate it in Section III on four SRSs with seeded omissions, showing the multi-LLM ensemble greatly boosts recall while preserving 95–100% suggestion plausibility. Section IV discusses these results, and Section V outlines future work.

II. Tool Architecture and Features

Our architecture is designed to balance structured extraction, semantic validation, and explainability, leveraging the complementary strengths of three LLMs. DeepSeek Chat has been shown to effectively extract structured elements and identify standardized patterns from formal requirements specifications [15], aligning with the findings of Jiang et al. [5]. GPT-4o Mini has demonstrated strength in analyzing quantitative requirements and detecting logical inconsistencies within requirement sets, employing techniques similar to those described by Liu et al. [10] and Kaur et al. [9]. Claude Sonnet 4 is noted for its strength in uncovering implicit requirements and contextual dependencies in complex specifications using a question-answering approach, as documented by Biswas et al. [3], and for adhering to explainable AI principles outlined by Sheh et al. [13]. By combining models with diverse capabilities, the tool follows an ensemble-like strategy: multiple LLM "opinions" are cross-checked and aggregated, analogous to a team of experts with different viewpoints working on the same problem to achieve more reliable results [1]. This approach is motivated by recent studies on multi-model AI systems, which have shown that aggregating outputs from several models can improve accuracy and reduce individual model biases [1]. In other words, this ensemble design mitigates the brittleness of single-model pipelines by ensuring that omissions missed by one model can be recovered by others.

Users can configure the tool by selecting any subset of these LLMs and choosing an aggregation method—majority voting, weighted voting, or a Meta-LLM fusion (where one selected LLM acts as a meta-model to interpret and reconcile the others' outputs). This flexibility allows analysts to tailor the analysis process based on their confidence in different models and the nature of the requirements. For example, majority or weighted voting harnesses the "wisdom of the crowd" effect to include requirements elements identified by most models, a

2643-1572/25/$31.00 ©2025 IEEE
DOI 10.1109/ASE63991.2025.00328

technique known to enhance reliability in ensemble learning [1]. The Meta-LLM fusion option is particularly useful when models disagree, since it resolves conflicts through a designated reconciling model rather than discarding potentially valuable insights. Figure 1 (left) depicts the tool's input configuration interface, where users upload requirement documents, select LLMs, and configure the aggregation strategy.

The tool features a modular architecture, comprising four primary components: (1) Input Processing, (2) Multi-LLM Analysis Engine, (3) Result Aggregation, and (4) Interactive Interface. A key aspect is the use of a prompt pipeline: a series of carefully crafted prompts orchestrates the flow of information through these components. Each phase of analysis is driven by specialized prompts that are fed in parallel to all selected LLMs, and the aggregated output of one phase automatically becomes the input to the next phase without user intervention. This design follows a structured chain of analysis methodology, ensuring that intermediate results (such as extracted requirements or generated domain models) seamlessly propagate through the workflow. These intermediate artifacts enhance transparency, allowing analysts to inspect how conclusions were reached.

All architectural and implementation files—including installation instructions, prompt templates, and component-specific algorithms—are available at https://doi.org/10.5281/zenodo.17037913.

1- Input Processing: Accepts direct text or SRS files (TXT/MD/DOCX/PDF). Uploaded documents undergo parallel extraction of requirements and context via structured prompts. Each selected LLM identifies requirement statements (with IDs) and extracts contextual data (overview, glossary, assumptions). Outputs are merged per the chosen aggregation strategy to yield a labeled requirements map and context set. Raw text input bypasses extraction and proceeds directly to analysis.

2-Analysis Engine: Executes three prompt-based tasks on the selected LLM(s), with per-stage aggregation:

- **Domain Model Generation**: Synthesizes entities, attributes, relationships, and a PlantUML diagram from requirements and context. Domain models provide a sound basis for completeness checks as argued in previous studies [4], [16]. When multiple LLMs are used, each generates a candidate model; these are merged via the aggregation method, ensuring comprehensive coverage.
- **External Completeness Checking**: Using the domain model and requirements set, the tool evaluates external completeness (whether the set of requirements covers all stakeholder goals and system contexts) by detecting missing functionality, domain concepts, associations, or attributes. Dedicated prompts guide each LLM to flag uncovered domain model elements. For example, the tool will flag any attribute or association defined in the domain model but not addressed by a requirement, as well as any typical user interaction in the domain that is missing from the specification. Likewise, if the domain model itself lacks an attribute or association that the requirements context implies, this omission is also reported as a potential missing requirement. The analysis targets gaps in functionality, data flows, and behavior typical in the domain. Each LLM then proposes candidate requirements to address each gap. This approach reflects expert practice and parallels AI-based assistants like ARIA [3]. When using multiple LLMs, suggestions are merged via the chosen aggregation method into a unified list. The result of this step is a set of proposed requirements that fill the identified external gaps.
- **Internal Completeness Checking**: In parallel with external checks, internal completeness is assessed for each requirement to ensure it is self-contained. For functional requirements, prompts guide the LLMs to verify essential elements—actor, action, object, and flag omissions. For non-functional requirements, the tool applies quality-scenario templates (stimulus, environment, system response, and response measure) to verify whether the statement encompasses all critical dimensions [2]. A completeness score is then computed, reflecting coverage of expected components, with brief explanations and improvement suggestions (e.g., "Requirement R5 does not specify the triggering condition"). Multiple LLM assessments are aggregated (majority, weighted, or meta-fusion) into a consensus score and recommendations, yielding a detailed list of incomplete requirements and suggested refinements.

3- Result Aggregation: Merges all outputs from the analysis engine into a single JSON structure containing the consensus domain model, external recommendations, and internal completeness annotations (scores and suggestions). This unified format supports UI visualization and programmatic export while ensuring consistent terminology across results.

4- Interactive Correction Interface: As shown in Figure 1 (right), the dashboard organizes results into panels for the generated domain model, detected requirement gaps with completeness scores, identified missing requirements, and suggested requirement statements accompanied by improvement rationales. Users accept, edit, or reject suggestions; accepted edits can optionally trigger re-analysis to update completeness assessments. This human-in-the-loop approach balances automated detection with expert validation, fostering trust and enabling continuous requirements improvement.

III. Evaluation Setup

To assess the tool's effectiveness, we designed an experiment on four distinct SRS documents from different domains: (1) a methane monitoring control system, (2) a smart-home system, (3) a patient-monitoring device, and (4) an Antarctic research station facility. These cover varied complexity and application areas. For each SRS, we followed the evaluation plan:

1) We identified a set of requirements "ranked as high and medium priority" in each original SRS and deliberately removed them. The deliberately removed requirements covered a mix of core functionality, actor interactions, and supporting quality constraints, ensuring that recall scores reflect omissions with both high and moderate

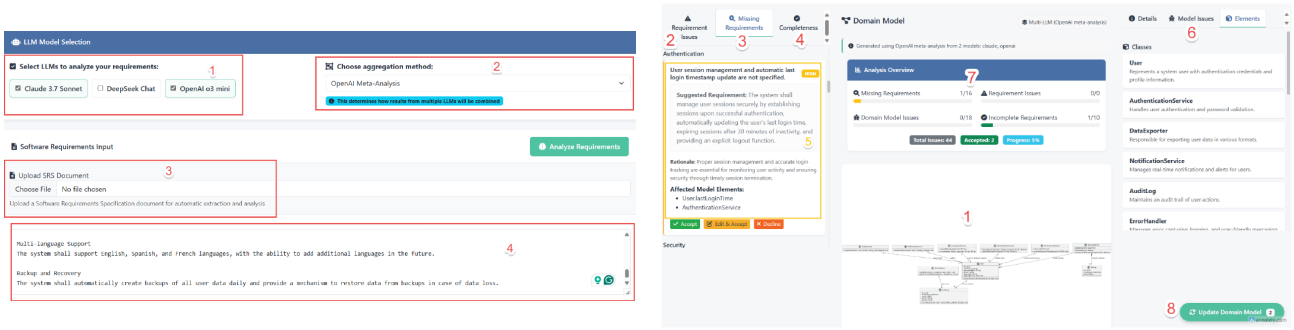

Fig. 1. **Left**: Input Configuration Interface showing (1) the LLM selection panel, (2) the aggregation method dropdown, (3) the file upload button, and (4) the requirements text area. **Right**: Analysis Dashboard displaying (1) the domain model visualization, (2) the general requirements issues panel, (3) the missing requirements panel, (4) completeness scores, (5) suggested requirement text with rationales, (6) domain model issues panel, (7) progress tracking metrics, and (8) update controls for requirements and domain model.

potential impact on the system model. For traceability, we recorded each removed item.

2) The tool was then executed under twelve configurations to assess single vs. ensemble performance:
   - **Single models:** Each LLM (DeepSeek, GPT-4o Mini, Claude Sonnet 4) in isolation for baseline recall and precision.
   - **Two-model ensembles:** Three pairings (DeepSeek+GPT, Claude+GPT, DeepSeek+Claude), tested with majority and meta fusion (GPT as meta).
   - **Three-model ensembles:** Full ensemble with majority voting, weighted voting, and meta fusion (GPT or Claude as meta). Weighted voting used normalized F_1-scores from single-model runs ($w_{\text{GPT}} = 0.45$, $w_{\text{Claude}} = 0.30$, $w_{\text{DeepSeek}} = 0.20$).

   Each configuration of the above 12 produced a list of suggested requirements for subsequent evaluation.
3) Measuring Recall: For each setting, we compared the tool's suggestions to the list of removed requirements. Any suggestion matching (or semantically equivalent to) a removed requirement is counted as a true positive. We compute recall = (correctly recovered) / (total removed). For example, if 5 requirements were removed and the tool suggested 4 of those, recall = 80
4) Assessing Plausibility (Precision): Suggestions not in the removed list are "extra" (possible false positives or new improvements). Lacking full ground truth, we conducted a guided plausibility check. Plausibility was measured via a *guided review* of all "extra" suggestions—i.e., those proposed by the tool but not matching any deliberately removed requirement. For each extra suggestion in every run:
   a) **Contextual Inspection:** We examined the SRS (overview, glossary, existing requirements, domain descriptions) to determine if any textual or conceptual basis existed for the suggestion.
   b) **Plausibility Criteria:** A suggestion was labeled *Plausible* if "all" the following conditions are met:
      - Used terminology or concepts mentioned elsewhere in the SRS,
      - Addressed an unhandled scenario or edge case implied by system goals,
      - Aligned with features commonly expected in that domain.

      It was labeled *Unlikely* if "any" of the following conditions are met:
      - Introduced out-of-scope functionality,
      - Duplicated existing requirements,
      - Lacked any textual or domain-knowledge support.
   c) **Dual Review:** Both authors independently judged each suggestion, then reconciled any disagreements through discussion.

   We computed the plausibility percentage as:

$$\frac{\text{\# of suggestions labeled "Plausible"}}{\text{Total \# of extra suggestions}} \times 100\%.$$

   This guided process serves as a proxy for precision, given the absence of an objective ground truth for extra suggestions.

All settings and results were recorded for analysis and are available at the evaluation repository https://doi.org/10.5281/zenodo.17037968. Recall was our primary metric, reflecting how well the tool finds known missing items. Precision (via plausibility) indicates how many suggestions were on-target rather than spurious. The evaluation was repeated for each SRS under the twelve configurations to gather comprehensive data.

IV. Results and Discussion

Table 1 summarizes the mean recall and plausibility scores across the four SRS documents, and the complete results for each execution on each SRS are available at the evaluation repository. Several clear patterns emerge:

Single-Model Performance. Individual LLMs exhibit both low average recall and high variability across domains. On Document 1, DeepSeek recovers only 17% of removed requirements versus 100% for GPT-4o Mini and Claude; on Document 2, recall ranges from 25%–50%; Document 3 drops

to 0%–20%; and Document 4 to 20%–40%. Overall, GPT-4o Mini and Claude average just 46.3% and 52.5% recall—despite 99%–100% suggestion plausibility—while DeepSeek averages 18.6%. Their standard deviations confirm that each model's strengths are domain-specific. This implies that relying on any single LLM for completeness checking is brittle and risks substantial coverage gaps.

Ensemble Benefits. Combining all three LLMs under the Meta-LLM fusion yields a substantial recall boost (80.6% with GPT as meta, and 75.2% with Claude as meta), demonstrating that each LLM uncovers distinct omissions and their union greatly expands coverage. Moreover, GPT-meta fusion cuts recall variability to σ =15.3% (Claude-meta: σ =11%). This implies that aggregation with Meta-LLM fusion not only maximizes recall but also ensures consistent performance across diverse SRS documents. By contrast, both majority voting (σ = 34.7%) and fixed-weight voting (σ = 30.3%) exhibit higher variability, underscoring their relative instability compared to the consistently robust Meta-LLM fusion.

Aggregation Strategy Matters: Meta-LLM fusion consistently outperforms both majority (61.25%) and static weighted voting (44.8% recall). For example, GPT-meta achieves a 35.8-point boost over weighted voting, highlighting the value of a learned reconciliation step over unweighted or fixed-weight schemes.

Two-Model Trade-Offs: Employing GPT-meta fusion with two-model ensembles (Claude+GPT at 78.6% recall and DeepSeek+GPT at 65.6% recall) provides a cost-effective compromise between computational overhead and coverage. In resource-constrained environments, selecting the top two models delivers performance close to the full ensemble while reducing runtime. Importantly, two-model Meta-LLM fusion substantially outperforms two-model majority voting—whose recall ranges only between 35.8% and 49.0%—because majority voting in this case includes a suggestion only when both models concur (e.g., if GPT proposes ten items, DeepSeek fifteen, and only five overlap, then only those five are accepted).

Plausibility and Precision. Across all configurations, over 95% of extra suggestions were judged plausible. For example, DeepSeek achieved just 20% recall on Document 4, yet all its suggestions were valid—few but highly relevant. High-recall ensembles also maintained strong plausibility (96–100%). On Document 3, Claude+GPT reached 80% recall with 89% plausibility; in contrast, GPT-only achieved 20% recall and 100% plausibility. This reflects a modest trade-off as coverage increases, but even 89% plausibility indicates strong utility. Plausibility scores remained stable across runs ($\sigma \leq 4.7\%$, most $\leq 2\%$), confirming consistent suggestion quality.

Model contributions: The data also highlight emerging results on each LLM's role. GPT-4o Mini was generally strong at logic/quantitative items (catching numeric or conditional requirements). Claude often caught contextual or high-level goals. DeepSeek worked well on formal or structured criteria. For example, Doc 1's gas-monitoring domain saw GPT/Claude pick up ventilation and alarm rules easily, while DeepSeek missed many specifics. By aggregating, the ensemble compensated for each model's blind spots.

TABLE I
AVERAGE RECALL AND SUGGESTION PLAUSIBILITY (± STD. DEV.)

Configuration	Recall (%)	σ Recall	Plaus. (%)	σ Plaus.
Single Models				
Claude Sonnet	52.5	34.0	100.0	0.0
GPT-4 Mini	46.3	36.8	99.0	2.0
DeepSeek Chat	18.6	15.4	98.8	2.5
Two-Model Ensembles				
Claude+GPT (meta)	78.6	8.7	95.6	4.7
DeepSeek+GPT (meta)	65.6	33.2	100.0	0.0
Claude+DeepSeek (maj.)	49.0	27.6	96.3	4.4
GPT+DeepSeek (maj.)	42.8	30.1	99.3	1.5
GPT+Claude (maj.)	35.8	36.6	98.2	2.2
Three-Model Ensembles				
All (GPT-meta)	80.6	15.3	99.2	1.7
All (Claude-meta)	75.2	11.0	98.2	2.1
All (maj.)	61.3	34.7	97.4	3.3
All (weighted)	44.8	30.3	98.3	3.5

V. CONCLUSION AND FUTURE WORK

Our evaluation on four diverse SRS documents shows that the ensemble consistently outperforms any single model—boosting recall by up to 3×—while maintaining very high plausibility of extra suggestions. The generated domain model provides a sound baseline for both external and internal completeness checks, and the interactive interface supports a practical human-in-the-loop refinement process. While promising, these findings are limited by the use of seeded omissions and modest-scale datasets, highlighting the need for broader validation on industrial SRSs.

Future Plans

To advance toward a production-grade tool, we will focus on three directions:

- **Large-Scale Benchmarking:** Benchmark on public SRS corpora (e.g., PROMISE, ReLiS) to statistically confirm performance across more diverse domains. We also plan controlled user studies with industry partners to measure productivity, error rates, and user satisfaction.
- **Adaptive Meta-LLM Fusion:** Develop a meta-fusion tuner that dynamically optimizes model weights or selects the best meta-model per domain, stabilizing recall across document types.
- **Scalability and RAG Integration:** Implement chunking and retrieval-augmented generation (RAG) to handle SRSs exceeding 1 MB while preserving cross-chunk context.
- **Conversational Elicitation.** We aim to close the loop between analysis and elicitation by embedding a conversational LLM agent. When gaps are detected, the agent will pose targeted questions to stakeholders, accelerating clarification and reducing manual back-and-forth. We will evaluate this conversational module's effect on requirement quality and elicitation efficiency.

REFERENCES

[1] Aisutra: Mitigating AI Hallucinations: The Power of Multi-Model Approaches. https://shorturl.at/ju0cP (2024), accessed: 2025-07-05

[2] Bass, L., Clements, P., Kazman, R.: Software architecture in practice. Addison-Wesley Professional (2021)

[3] Biswas, C., Das, S.: ARIA-QA: AI-agent based requirements inspection and analysis through question answering. Innovations in Systems and Software Engineering pp. 1–16 (2024)

[4] Chattopadhyay, A., Malla, G., Niu, N., Bhowmik, T., Savolainen, J.: Completeness of natural language requirements: A comparative study of user stories and feature descriptions. In: 2023 IEEE 24th International Conference on Information Reuse and Integration for Data Science (IRI). pp. 52–57. IEEE (2023)

[5] Jiang, Q., Gao, Z., Karniadakis, G.E.: Deepseek vs. chatgpt vs. claude: A comparative study for scientific computing and scientific machine learning tasks. Theoretical and Applied Mechanics Letters **15**(3), 100583 (2025)

[6] Jurkiewicz, J., Nawrocki, J., Ochodek, M., Głowacki, T.: Hazop-based identification of events in use cases: An empirical study. Empirical Software Engineering **20**, 82–109 (2015)

[7] Kassab, M., AbdElhameed, M.: Mapping the landscape of requirements completeness: Definitions, techniques, tools, and the emerging role of AI. In: Proceedings of the 2025 Euromicro Conference on Software Engineering and Advanced Applications (SEAA). IEEE (2025)

[8] Kassab, M., Laplante, P.: The current and evolving landscape of requirements engineering in practice. IEEE Software **39**(5), 76–83 (2022)

[9] Kaur, K., Singh, P., Kaur, P.: A review of artificial intelligence techniques for requirement engineering. Computational Methods and Data Engineering: Proceedings of ICMDE 2020, Volume 2 pp. 259–278 (2020)

[10] Liu, S., Chen, X., Jin, Z., Zhang, M.: Automated inconsistency analysis of real-time requirements: A domain expert friendly approach. In: 2022 IEEE 24th Int Conf on High Performance Computing & Communications; 8th Int Conf on Data Science & Systems; 20th Int Conf on Smart City; 8th Int Conf on Dependability in Sensor, Cloud & Big Data Systems & Application (HPCC/DSS/SmartCity/DependSys). pp. 1109–1114. IEEE (2022)

[11] Menzel, I., Mueller, M., Gross, A., Doerr, J.: An experimental comparison regarding the completeness of functional requirements specifications. In: 2010 18th IEEE International Requirements Engineering Conference. pp. 15–24. IEEE (2010)

[12] Mirabelli, V., Eckman, H.: Scale up your requirements management practices. Blueprint, Info-Tech Research Group (September 2024), https://shorturl.at/PYLSg, last revised September 26, 2024

[13] Sheh, R., Monteath, I.: Defining explainable ai for requirements analysis. KI-Künstliche Intelligenz **32**(4), 261–266 (2018)

[14] The Standish Group International: CHAOS Report: Decision Latency Theory. https://www.standishgroup.com/products/project-resolution-benchmark (2018), accessed: 2025-07-08

[15] Zhi, J.: LLM-Ensemble: A Simple Framework for Ensembling Large Language Models. https://junchenzhi.github.io/LLM-Ensemble/ (2024), accessed: 2025-07-05

[16] Zowghi, D., Gervasi, V.: On the interplay between consistency, completeness, and correctness in requirements evolution. Information and Software technology **45**(14), 993–1009 (2003)