ReqLens: An Interactive Evidence-Grounded Multi-Agent System for Document-to-Requirements Generation

Manojit Chakraborty
Bosch Research & Technology Center, Bangalore
Manojit.Chakraborty@in.bosch.com

Pranav Laddhad
IIIT Bangalore, India
pranav.laddhad2@gmail.com

Sagnik Ghosh
IIT Mandi, India
sagnik4998@gmail.com

Abstract

Large language models can transform project documents into software requirements, but direct generation may introduce unsupported claims and hidden inconsistencies. We present ReqLens, an interactive multi-agent NLP system that extracts candidate requirements from mixed project documents, links them to source evidence, exposes graph-based dependencies and conflicts, and generates a Software Requirements Specification (SRS) through a human-in-the-loop workflow. We evaluate ReqLens on 13 projects using direct-generation and modular baselines, as well as component ablations. ReqLens achieves the strongest overall balance across unsupported-requirement rejection, valid-requirement retention, and seeded-defect filtering. A formative study with four domain experts provided qualitative feedback on the system's interface and review workflow. ReqLens is accompanied by code, data, an installable package, and a narrated screencast.

1 Introduction

Project teams often need to transform heterogeneous natural-language artifacts, including stakeholder interviews, meeting notes, user stories, legacy specifications, policies, test cases, and change requests into structured requirements that are complete, traceable, and reviewable. Large language models (LLMs) are well suited to support this document-to-requirements workflow because they can summarize long inputs, extract candidate requirements, rewrite informal statements, and generate specification text. However, direct LLM-based generation remains difficult to trust in requirements engineering: generated requirements may be plausible but unsupported by source evidence, quality attributes may be over-generalized and dependencies among requirements, policies, tests and change requests may remain implicit.

We present REQLENS, an interactive NLP system for evidence-grounded requirements extraction, verification, review and SRS generation. ReqLens follows the principle: *LLMs propose, evidence gates, graphs validate, and humans approve*. Rather than using an LLM as an autonomous specification writer, ReqLens decomposes the workflow into typed intermediate steps: candidate requirement extraction, source-span evidence verification, quality analysis, dependency construction, traceability generation, human review, and controlled SRS composition. Each accepted requirement is linked to source evidence, review status, quality findings, and graph-based relations to other project artifacts.

The contribution of ReqLens is not another language model, but a deployable control architecture for grounded, human-reviewable generation from mixed project documents. Our contributions are:

- Evidence-gated multi-agent architecture that converts free-form LLM outputs into source-linked, reviewable requirements;
- Requirements graph supporting source link, dependency, duplicate, conflict, traceability, and change-impact inspection;
- Interactive human-review workflow for requirement approval and controlled SRS generation; and
- Evaluation against direct generation and modular baselines, including component ablations and a formative study with four domain experts.

2 Related Work

Grounded generation and factuality. Retrieval-augmented generation grounds model outputs in external evidence (Lewis et al., 2020; Gao et al., 2023), yet retrieved context does not guarantee factual support. Prior work shows that fluent generations may still be unsupported or inconsistent

Proceedings of the 15th International Joint Conference on Natural Language Processing and the 5th Conference of the Asia-Pacific Chapter of the Association for Computational Linguistics, pages 74–81
November 6-10, 2026 ©2026 Association for Computational Linguistics

with their sources (Maynez et al., 2020; Ji et al., 2023; Manakul et al., 2023). ReqLens therefore verifies each generated requirement against identifiable source spans and routes contradicted or insufficiently supported candidates to human review.

Agentic and human-in-the-loop NLP. Agentic systems decompose complex tasks into specialized components for retrieval, reasoning, tool use, and generation (Yao et al., 2023; Wu et al., 2023). ReqLens adopts this pattern but emphasizes inspectable, typed intermediate artifacts - including requirements, evidence labels, quality findings, graph relations, and clarification questions - that reviewers can revise before SRS generation.

NLP for requirements engineering. ReqInOne is the closest LLM-based system, using modular prompting for requirement extraction, classification, and SRS generation (Zhu et al., 2025). Earlier tools such as QuARS (Fabbrini et al., 2001) and SREE (Tjong and Berry, 2013) rely primarily on rules or lexicons for quality analysis, while platforms such as DOORS (Rundlet and Miller, 1994) and ReqView (req, 2015) focus on managing authored requirements. ReqLens instead combines derivation from mixed natural-language artifacts with source-span verification, graph-based consistency analysis, and human approval.

3 System Design

ReqLens transforms heterogeneous project artifacts into evidence-grounded, reviewable software requirements. It targets requirements engineers, product owners, QA engineers, compliance reviewers, and researchers working with stakeholder notes, user stories, policies, legacy specifications, test cases, and change requests.

Fig. 1 shows the architecture. Agents exchange schema-validated intermediate artifacts over a shared evidence store, requirements graph, and review database. A common LLM gateway manages prompt versions, retries, logging, and output validation.

System novelty: The novelty of ReqLens lies in its control architecture rather than in a new language model. First, requirement generation and evidence verification are performed by separate agents, preventing a model's own generation from serving as its evidence. Second, requirements are validated jointly through a typed graph, exposing conflicts, duplicates, dependencies, and change impacts that

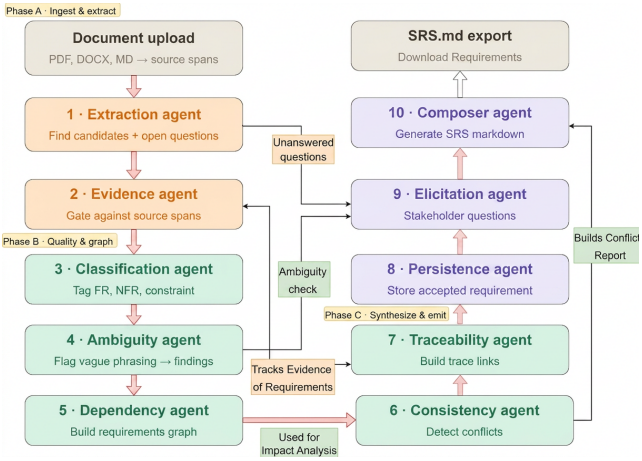


Figure 1: ReqLens multi-agent architecture.

cannot be detected from isolated candidates. Third, temporary model outputs are separated from the approved project state: only human-approved requirements and graph relations can be used for final SRS generation.

3.1 Source Processing and Evidence Grounding

Uploaded PDF, DOCX, Markdown, CSV, and text artifacts are segmented into paragraphs, clauses, user stories, or speaker turns. Each source span receives a stable identifier and metadata such as document, section, speaker, page, and character offsets, and is indexed for retrieval.

The **Extraction Agent** converts these spans into atomic candidate requirements and unresolved questions. The **Evidence Agent** verifies each candidate against retrieved source spans, labeling it as ENTAILED, CONTRADICTED (if a source span explicitly refutes the candidate), or INSUFFICIENT (lacking evidence). Only entailed candidates proceed normally, others are flagged for review. This strict grounding ensures that unsupported claims are not present in the requirement set.

3.2 Requirement Analysis and Graph Validation

The **Classification Agent** labels supported candidates as functional, non-functional, business rules, domain assumptions or constraints. The **Ambiguity Agent** flags vague wording, weak modality, non-atomic statements, and unverifiable conditions.

The **Dependency Agent** proposes typed relations such as `derived_from`, `refines`, `requires`, `conflicts_with`, `duplicates`, `constrains`, `tested_by` etc. Candidate pairs are selected through semantic retrieval and graph neighbor-

Table 1: ReqLens agents and their typed intermediate artifacts.

Agent	Input	Output	Responsibility
Extraction	Source spans	Requirements, open questions	Extracts atomic requirements and links them to source spans.
Evidence	Candidate and retrieved spans	ENTAILED, CONTRADICTED, or INSUFFICIENT	Checks whether the candidate is supported by the source.
Classification	Supported requirement	Requirement type	Labels it as `functional`, `non-functional`, `business rules`, `domain assumptions` or `constraints`.
Ambiguity	Requirement text	Quality findings	Flags vague, weak, non-atomic, or unverifiable statements.
Dependency	Requirement pairs	Candidate graph edges	Proposes relations such as `derived_from`, `refines`, `requires`, `conflicts_with`, `duplicates`, `constrains`, `tested_by` etc.
Consistency	Requirements graph	Conflict and duplicate warnings	Detects `overlaps`, `contradiction`, `duplicates` and `inconsistencies`
Traceability	Accepted requirements and evidence	Traceability links	Connects requirements to sources and related artifacts.
Persistence	Review decisions	Approved project state	Stores accepted requirements and approved graph links.
Elicitation	Questions and quality findings	Clarification questions	Converts unresolved findings into stakeholder questions.
Composer	Approved requirements and graph	SRS and reports	Generates the SRS from approved, evidence-supported content.

hoods. The **Consistency Agent** then validates the resulting graph for duplicate clusters, contradictions, unsupported nodes, and incompatible constraints. Thus, Dependency proposes graph structure, whereas Consistency checks whether that structure exposes defects.

For change-impact analysis, ReqLens retrieves requirements matching a change request and traverses selected graph relations for up to two hops. The interface distinguishes directly matched requirements from indirectly connected impacts.

3.3 Review and Controlled Generation

The **Traceability Agent** connects accepted requirements to source spans and related policies, tests, changes, and dependent requirements. Reviewers can accept, reject, revise, or defer each candidate after inspecting its evidence and analysis results. The **Persistence Agent** stores only approved requirements and graph relations, keeping temporary LLM outputs separate from the reviewed project state.

The **Elicitation Agent** converts unresolved questions, ambiguity findings, and conflicts into stakeholder-facing clarifications. Finally, the **Composer Agent** generates the SRS only from approved, evidence-supported requirements and validated graph context. Rejected, contradicted, insufficient, or deferred candidates are excluded from the main specification.

3.4 Implementation Details

ReqLens is implemented with a Streamlit frontend and a FastAPI backend. Source spans and their embeddings are maintained in the evidence store, while requirements and typed relations are persisted in the graph store. Agent outputs are exchanged as schema-validated JSON objects containing stable identifiers, labels, confidence values, evidence references, and short explanations.

In the reported experiments, ReqLens uses an Azure OpenAI `GPT-5.5-mini` deployment for agent execution and `text-embedding-3-large` for source-span retrieval. The Evidence Agent retrieves the top-$k = 3$ spans for each candidate. The deployed model endpoint used in our experiments did not expose a configurable temperature parameter; all runs therefore use the provider default. Exact prompts, schemas, model settings, and installation instructions are included in the released artifact.

4 Interface and Demonstration

ReqLens is demonstrated as an interactive web application than a one-shot SRS generator. Its interface exposes candidate requirements, source evidence, quality findings, graph relations, and review decisions, allowing users to inspect how each requirement was produced before it enters the approved project state. The system is accompanied

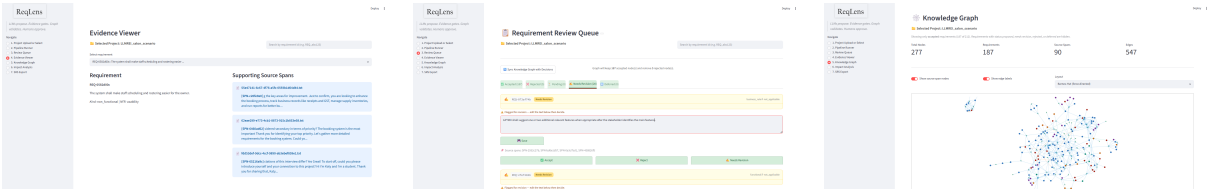


Figure 2: Principal ReqLens interaction views for evidence inspection, requirement review, and graph-based analysis.

by an installable package and a narrated screencast.

The demonstration uses a salon management project containing stakeholder notes, user stories, legacy requirements, test cases, and change requests. The workflow consists of four stages:

1. **Project setup and agent execution:** Users upload or select project artifacts and run the multi-agent pipeline. The Pipeline Runner displays the execution status and outputs of individual agents rather than hiding the workflow behind a single generation request.

2. **Evidence inspection and review:** For each candidate requirement, the Evidence Viewer shows its evidence label and the supporting or contradicting source spans. The Review Queue allows the candidate to be accepted, rejected, revised, or deferred.

3. **Graph and impact analysis:** Approved requirements are displayed with typed relations such as dependencies, constraints, duplicates, and conflicts. Users can inspect individual relations or submit a change request to identify directly and indirectly affected requirements.

4. **Controlled generation and export:** ReqLens generates the SRS from the reviewed project state and exports its traceability matrix, graph representation, conflict report, and unresolved clarification questions.

Fig. 2 illustrates the three central review views. The Evidence Viewer enables users to verify whether a generated requirement is supported by the project documents. The Review Queue combines evidence, quality findings, and graph warnings in a single decision view. The Knowledge Graph provides project-level context that is unavailable when requirements are reviewed independently. The complete workflow screenshots are provided in Appendix B.

A representative demonstration case begins with two requirements where one says booking system to be available through "both an app and a website," while other specifies a "website-only" solution and explicitly forbids a mobile app. The Dependency Agent gives `conflicts_with` edge between these requirements. The Consistency Agent marks former candidate requirement as `CONTRADICTED`. To resolve this conflict, the reviewer is prompted with stakeholder clarification questions; the candidate requirement is only accepted once these clarifications are resolved.

This interaction demonstrates the main technological distinction of ReqLens: model outputs are not treated as final requirements. They remain inspectable proposals whose evidence, cross-requirement effects, and human review status determine whether they can influence the final specification.

5 Evaluation

We evaluate both the automated safeguards in ReqLens and the usefulness of its human-review interface. The evaluation addresses three research questions:

RQ1 How effectively does ReqLens reject unsupported requirements while retaining source-supported requirements?

RQ2 How much does graph-based consistency checking reduce the retention of seeded duplicates and contradictions?

RQ3 How do domain experts assess the usefulness, navigability, review support, and efficiency of the ReqLens interface?

5.1 Benchmark and Systems

We evaluate ReqLens on 13 projects (7 PROMISE, 6 PURE) derived from PURE (Ferrari et al., 2017) and PROMISE (C.H. Jane and Dan, 2007). Because these datasets primarily contain structured requirements, we enrich each project with exactly 3 synthetic elicitation artifacts (39 total), including

Table 2: Benchmark statistics across 13 projects. Mean and range are computed per project.

Item	Total	Mean	Min–Max
Candidate requirements	650	50.0	17–107
Gold requirements	544	41.9	12–92
Unsupported seeds (Track 1)	106	8.2	5–15
Poison rate (Track 1 %)	–	16.3	13.6–29.4
Contradiction seeds (Track 2)	65	5.0	5
Duplicate seeds (Track 2)	65	5.0	5

meeting notes, email threads and interview transcripts. Table 2 reports the resulting benchmark statistics.

The benchmark contains two controlled tracks. In the **evidence-grounding track** (Track 1), systems receive the same project documents and a shuffled pool containing source-supported gold requirements and plausible but unsupported candidate seeds lacking direct evidence in any source artifact (16.3% average poison rate). Each system must determine which candidates should enter the accepted requirement set. In the **defect-awareness track** (Track 2), a set of 5 contradictions and 5 semantic duplicates are seeded into the source artifacts before requirement extraction. We then determine whether these defects remain represented in the resulting requirement set. We compare five configurations:

- **Direct LLM**: a single prompt operating on the project artifacts;
- **ReqInOne-style**: modular extraction and classification without an evidence gate or graph-based consistency checking;
- **ReqLens w/o Evidence**: the full pipeline with the Evidence Agent disabled;
- **ReqLens w/o Consistency**: the full pipeline with the Consistency Agent disabled; and
- **ReqLens**: the complete evidence-gated and graph-validated system.

All automatic configurations are evaluated before human intervention so that the contributions of the system components are not confounded by reviewer corrections. Where applicable, the systems use the same underlying language model, input artifacts, and decoding budget; the ablations differ only in the disabled component.

5.2 Metrics and Judgment Protocol

We report three intuitive, higher-is-better metrics derived from the original error rates.

- **Unsupported Rejection** is $1 - \mathrm{UAR}$, where UAR is the fraction of unsupported candidates incorrectly accepted.
- **Valid Retention** is $1 - \mathrm{GRR}$, where GRR is the fraction of source-supported requirements incorrectly rejected.
- **Defect Filtering** is $1 - \mathrm{DLR}$, where DLR is the fraction of seeded duplicates or contradictions retained in the final requirement set.

Details about original error-rate definitions and defect-type breakdowns are provided in our GitHub repository. Metrics are computed independently for each project. All results are averaged over 5 independent runs per project. We first compute the average metric within each project and then report the macro-average and standard deviation across all 13 evaluation projects. This prevents projects with more artifacts or requirements from dominating the results.

Unsupported candidates and seeded defects are tracked through deterministic registries. An LLM judge is used only for cases in which semantic equivalence or defect retention cannot be resolved through registry matching. The judge follows a fixed rubric, receives the relevant source evidence and candidate requirements, and is blind to system identity. Its prompt, configuration, and manually validated sample are included in the released evaluation artifacts.

5.3 Automatic Results

Table 3 reports the automatic evaluation. Percentages are macro-averages across projects; higher values across all 3 columns indicate better performance.

Evidence grounding. ReqLens rejects 99.2% of unsupported candidates while retaining 97.0% of source-supported requirements. Direct LLM achieves a slightly higher unsupported-rejection score of 99.4%, but retains only 93.7% of valid requirements, indicating that its result is partly due to over-rejection. While Direct LLM is better at rejecting unsupported candidates, ReqLens achieves a superior overall balance.

The no-Evidence ablation shows the opposite failure mode: it rejects none of the unsupported candidates while retaining all valid requirements, because every candidate is accepted without source verification. The two metrics must therefore be

Table 3: Automatic evaluation across 13 projects and five runs. Bold denotes the full ReqLens system; † denotes the numerical maximum in each column. Values are macro mean ± standard deviation; higher is better.

System	Unsupported rejection	Valid retention	Defect filtering
Direct LLM	$99.4 \pm 0.3^{\dagger}$	93.7 ± 0.2	59.4 ± 4.4
ReqInOne-style	88.0 ± 20.8	90.9 ± 4.5	66.3 ± 4.8
ReqLens w/o Evidence	0.0 ± 0.0	$100.0 \pm 0.0^{\dagger}$	$74.8 \pm 3.9^{\dagger}$
ReqLens w/o Consistency	99.0 ± 0.3	97.2 ± 0.2	58.3 ± 3.6
ReqLens	$\mathbf{99.2 \pm 0.0}$	$\mathbf{97.0 \pm 0.2}$	$\mathbf{72.5 \pm 1.2}$

interpreted jointly. Compared with the ReqInOne-style baseline, ReqLens improves both unsupported rejection and valid retention while showing much lower variance across runs.

Graph consistency. ReqLens filters 72.5% of seeded duplicates and contradictions (with the detailed breakdown of these categories available in our repository), compared with 59.4% for Direct LLM and 66.3% for the ReqInOne-style baseline. Removing the Consistency Agent lowers defect filtering to 58.3%, close to the Direct LLM result, showing that explicit cross-requirement analysis is the main contributor to defect control.

The no-Evidence ablation reaches a slightly higher defect-filtering score of 74.8%, but its larger variance and complete failure on unsupported rejection make it unsuitable as a reliable configuration. Overall, the results confirm that the Evidence Agent and Consistency Agent address complementary failure modes, and that full ReqLens offers the strongest balance across grounding, retention, and defect filtering.

5.4 Formative Expert Evaluation

Four independent domain experts used ReqLens to review a fixed set of candidate requirements. They inspected source evidence, navigated agent outputs, and recorded accept, reject, revise, or defer decisions. After the task, each expert rated four aspects of the system on a five-point Likert scale, where 1 denotes the lowest and 5 the highest usefulness or perceived correctness. Given the small sample, we report medians and ranges rather than significance tests. The study is intended as a formative assessment of the implemented interface, not as a population-level usability study.

Requirement-review support received the highest median rating 4.75, while agentic navigation, and perceived correctness each received a median of 4.5. The overall UI usefulness received a median of 4.25, indicating there is still scope for improvement. Taken together, these ratings provide encouraging evidence that experts found ReqLens useful and practical for requirement-level review. The consistently high scores support the usability of the evidence-guided workflow, while a larger controlled study is needed to quantify its effect on review accuracy and efficiency.

Table 4: Formative expert evaluation ($n = 4$). Likert measures use a 1–5 scale; higher is better.

Measure	Median	Min-Max
Overall tool/UI usefulness	4.25	4–5
Agentic navigation usefulness	4.5	4–5
Requirement-review usefulness	**4.75**	4–5
Perceived review correctness	4.5	4–5

6 Availability and Reproducibility

ReqLens is available at https://github.com/pld4kor-dot/ReqLens under the MIT License. The repository includes the application, Docker setup, prompts, schemas, demo projects, datasets, data builder, and evaluation scripts. The narrated demonstration video is available at https://youtu.be/B_LSZfwBgU4.

The released configuration records the models, retrieval settings, graph backend, and software versions used in the experiments. Agent outputs and review decisions are stored as typed relational records in PostgreSQL. Running ReqLens requires user-provided model API credentials.

7 Limitations

ReqLens is designed as a decision-support tool rather than a fully autonomous requirements authoring system. Its evidence and graph checks improve reviewability, but difficult cases remain when support is implicit across several sources, or expressed through domain-specific terminology. Such cases still require expert interpretation.

The evaluation combines 13 controlled project benchmarks with a formative study involving four domain experts. The synthetic elicitation artifacts enable reproducible stress testing of unsupported requirements, duplicates, and contradictions, while the expert study provides initial evidence of practical usefulness. Broader validation on larger industrial datasets with more users remains an important next step. For regulated or safety-critical settings, ReqLens should therefore be used to assist expert review rather than to replace it.

References

2015. Reqview. www.reqview.com.

L. Huang C.H. Jane, M. Sepideh and P. Dan. 2007. Promise dataset. In *Available: https://zenodo.org/records/268542*.

Fabrizio Fabbrini, Mario Fusani, Stefania Gnesi, and Giuseppe Lami. 2001. The linguistic approach to the natural language requirements quality: benefit of the use of an automatic tool. In *proceedings 26th annual NASA Goddard software engineering workshop*, pages 97–105. IEEE.

Alessio Ferrari, Giorgio Oronzo Spagnolo, and Stefania Gnesi. 2017. Pure: A dataset of public requirements documents. In *2017 IEEE 25th international requirements engineering conference (RE)*, pages 502–505. IEEE.

Yunfan Gao, Yun Xiong, Xinyu Gao, Kangxiang Jia, Jinliu Pan, Yuxi Bi, Yi Dai, Jiawei Sun, Meng Wang, and Haofen Wang. 2023. Retrieval-augmented generation for large language models: A survey. *arXiv preprint arXiv:2312.10997*.

Ziwei Ji, Nayeon Lee, Rita Frieske, Tiezheng Yu, Dan Su, Yan Xu, Etsuko Ishii, Ye Jin Bang, Andrea Madotto, and Pascale Fung. 2023. Survey of hallucination in natural language generation. *ACM Computing Surveys*, 55(12):1–38.

Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, et al. 2020. Retrieval-augmented generation for knowledge-intensive nlp tasks. *Advances in neural information processing systems*, 33:9459–9474.

Potsawee Manakul, Adian Liusie, and Mark Gales. 2023. SelfCheckGPT: Zero-resource black-box hallucination detection for generative large language models. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 9004–9017, Singapore. Association for Computational Linguistics.

Joshua Maynez, Shashi Narayan, Bernd Bohnet, and Ryan McDonald. 2020. On faithfulness and factuality in abstractive summarization. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 1906–1919, Online. Association for Computational Linguistics.

Nancy Rundlet and William D. Miller. 1994. Requirements management: Doors to the battlefield of the future. *INCOSE International Symposium*, 4(1):59–66.

Sri Fatimah Tjong and Daniel M Berry. 2013. The design of sree—a prototype potential ambiguity finder for requirements specifications and lessons learned. In *International Working Conference on Requirements Engineering: Foundation for Software Quality*, pages 80–95. Springer.

Qingyun Wu, Gagan Bansal, Jieyu Zhang, Yiran Wu, Shaokun Zhang, Erkang Zhu, Beibin Li, Li Jiang, Xiaoyun Zhang, and Chi Wang. 2023. AutoGen: Enabling next-gen LLM applications via multi-agent conversation framework. *arXiv preprint arXiv:2308.08155*.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. 2023. ReAct: Synergizing reasoning and acting in language models. In *The Eleventh International Conference on Learning Representations*.

Taohong Zhu, Lucas C Cordeiro, and Youcheng Sun. 2025. Reqinone: A large language model-based agent for software requirements specification generation. In *2025 IEEE 33rd International Requirements Engineering Conference (RE)*, pages 449–457. IEEE.

A Evaluation Details

Metric definitions: Let N_u denote the number of unsupported candidates, N_g the number of source-supported reference requirements, and N_d the number of seeded duplicates and contradictions. Let N_u^{acc}, N_g^{rej}, and N_d^{ret} denote the corresponding accepted, rejected, and retained counts. The original error-rate metrics are:

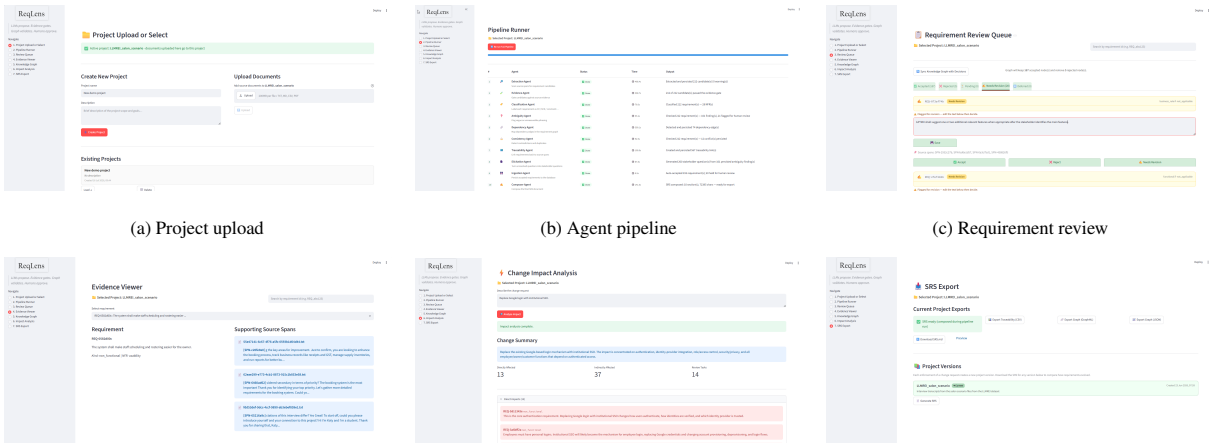
$$\text{UAR} = \frac{N_u^{\text{acc}}}{N_u}, \quad \text{GRR} = \frac{N_g^{\text{rej}}}{N_g}, \quad \text{DLR} = \frac{N_d^{\text{ret}}}{N_d}. \tag{1}$$

The main paper reports their more intuitive complements: Unsupported Rejection $= 1 - \text{UAR}$, Valid Retention $= 1 - \text{GRR}$, and Defect Filtering $= 1 - \text{DLR}$. All three headline metrics are therefore higher-is-better.

Formative expert study: Four independent experts reviewed a fixed set of requirements using ReqLens. They inspected source evidence, navigated agent outputs, and made accept, reject, revise, or defer decisions. Ratings used a five-point Likert scale, where 1 denotes "not useful" and 5 denotes "very useful." For perceived review correctness, 1 denotes "not accurate" and 5 denotes "very accurate." Table 5 reports the individual ratings; medians and ranges are reported in the main paper.

B Demonstration Workflow

The demo workflow uses a salon management project containing stakeholder notes, user stories, policy constraints, test cases, and change requests. Fig. 3 shows the end-to-end interaction: project



(a) Project upload (b) Agent pipeline (c) Requirement review

(d) Evidence inspection (e) Change-impact analysis (f) SRS and artifact export

Figure 3: End-to-end ReqLens workflow. Users upload project artifacts, inspect agent execution and source-span, review candidate requirements, analyze change impacts, and export approved SRS and traceability artifacts.

Table 5: Individual expert ratings. Likert scale.

Measure	E1	E2	E3	E4
Tool/UI usefulness	4	4	4	5
Agentic navigation	4	4	5	5
Requirement-review usefulness	4	5	5	5
Perceived review correctness	5	4	4	5

setup, agent execution, evidence inspection, human review, change-impact analysis, and SRS export.

Interaction sequence: Users first upload or select a project and run the multi-agent pipeline over the available stakeholder notes, user stories, policies, test cases, and change requests. The system then presents the extracted candidate requirements together with their evidence labels, supporting or contradicting source spans, ambiguity findings, and proposed graph relations. For each candidate, reviewers can accept, reject, revise, or defer the requirement while inspecting the rationale produced by the relevant agents. Approved requirements and validated graph relations are synchronized into the reviewed project state and become available for dependency exploration, conflict inspection, and change-impact analysis. Finally, ReqLens generates the SRS only from this approved state and exports the corresponding traceability matrix, graph representation, conflict report, and unresolved clarification questions.

B.1 Representative Decision Trace

Table 6 illustrates how ReqLens handles a contradictory requirement pair. In this example, one source artifact states that the booking system should be available through both an app and a website, while another states that booking should be website-only.

Table 6: Representative evidence-gated decision trace.

Stage	Artifact or decision
Source	Conflicting needs: booking through both app and website versus website-only booking.
Dependency	A `conflicts_with` relation is proposed between the two requirements.
Consistency	The pair is flagged as inconsistent and routed for review.
Resolution	Reviewer inspects the evidence and may request stakeholder clarification.
Output	The unresolved conflict is excluded from the SRS until clarification or revision.

This trace demonstrates how ReqLens separates candidate generation from project-state modification. Both requirements remain visible for inspection, but neither conflicting interpretation is silently promoted as final. The conflict edge preserves the reason for the warning, while the review state records whether the issue was rejected, revised, deferred, or sent for clarification.

C Ethical Considerations

The public ReqLens artifact uses only public and synthetically generated requirements data. Industrial deployments may involve confidential business information or personal data contained in stakeholder artifacts. Such deployments should include access control, redaction, secure storage, and clear logging practices for prompts, model outputs, and review decisions. ReqLens is designed to keep final requirement acceptance under human control rather than delegating approval to an LLM.