Towards Improving the Reliability of LLMs in Requirements Engineering with Structured Confidence and Tag Governance

***Abstract*—Automating requirements engineering with large language models (LLMs) remains risky: existing tools frequently hallucinate tags and lack field-level confidence, making their outputs unreliable for downstream agile planning. To address this, we propose a scalable, domain-flexible pipeline that first parses stakeholder prose into *High-Level JSON* (HLJ), a structured, confidence-annotated schema introduced here, and then applies a versioned tag governance stage for enrichment and validation. Benchmarked on 30 real-world FinTech and SaaS requirements, our evaluation shows that Opus4 and Meta-70B outperform GPT-4.1 by over 0.31 F1 points in our strictest tag governance stage, while our pipeline substantially improves tag precision across all models (e.g., for GPT-4.1: from 0.657 to 0.897). By open-sourcing our scripts, data, and evaluation harness, we aim to make robust, auditable requirement parsing reproducible across domains. Future work will extend our approach to larger, more diverse datasets and enable interactive, human-in-the-loop agile task refinement.**

***Index Terms*—Requirements engineering, large language models, JSON validation, tag governance, benchmark evaluation, Agile automation, FinTech, SaaS**

I. Introduction

Modern software projects, especially at agile scale or in regulated domains such as Fintech and SaaS are critically dependent on the quality of their requirements, where automation of requirements generation could potentially improve the quality of requirements. However, automating the tasks of requirements engineering with large language models (LLMs) remains challenging. State-of-the-art LLMs are prone to hallucinating tags, introducing irrelevant or misleading labels, and rarely provide field-level confidence scores, transparent audit trails, or robust versioning for traceability [1], [2]. These shortcomings directly threaten downstream workflows, resulting in misprioritized tasks, inconsistent project metadata, and increased compliance risks.

For organizations where traceability and correctness are essential, relying on "black-box" AI outputs is unacceptable [1]. Project managers and engineers demand requirement artifacts that are not only plausible, but verifiable, with every tag, confidence score, and breakdown step being open to audit and review [2]. Existing LLM-powered tools too often neglect these demands, leaving teams to manually repair or reinterpret outputs.

To address these challenges, we propose a modular, benchmark-driven pipeline for hallucination-resistant requirement parsing and structured output evaluation. Our approach parses stakeholder prose into High-Level JSON (HLJ) through providing confidence-scored and structured representations of requirements and applies a versioned tag governance layer for enrichment and validation. By benchmarking multiple leading LLMs across two domains and releasing open datasets, scripts, and audit logs, we aim to set a new standard for transparency and reliability in requirements engineering automation [3], [4].

Our main contributions are as follows:

- **A modular pipeline** for converting raw requirements to confidence-scored, auditable HLJ representations;
- **A domain-flexible, versioned tag governance system** for robust tag enrichment, validation, and deduplication;
- **A reproducible benchmarking harness** to compare multi-model LLM performance on real-world FinTech and SaaS data;
- **A fully open dataset and toolchain** supporting downstream Agile and enterprise planning workflows;
- **Comprehensive analysis and lessons** for future research on interactive, human-in-the-loop requirement refinement.

Motivating Example: Why Reliable Tagging Matters

Consider a team using an LLM tool for a FinTech API. The model suggests:

```
["api", "rest", "compliance", "currency-conversion"]
```

But `currency-conversion` does not appear in the requirement, while a critical tag like `openbanking` is missing. There's also no way to know which tags are reliable or how they were selected—making review a pain.

With our pipeline:

```
[
  {"tag": "api", "confidence": 0.95},
  {"tag": "openbanking", "confidence": 0.80},
  {"tag": "currency-conversion", "confidence": 0.10,
       "provenance": "dropped"}
]
```

Now, every tag is confidence-scored and fully auditable. This makes review smoother and automation safer—so teams can actually trust what the AI outputs.

II. Proposed Framework and Methodology

To enable reliable, auditable, and reproducible requirements engineering automation, we propose a modular, multi-model pipeline that supports fine-grained traceability from raw requirements to structured outputs. Our approach combines advances in LLM-based parsing, multi-stage tag governance,

and comprehensive audit logging, supporting both research rigor and industrial deployment. Below, we detail the design principles, system evolution, and technical workflows that underpin our framework.

A. Design Rationale

Our pipeline is guided by three core principles: (1) **modularity**, (2) **multi-model benchmarking**, and (3) **end-to-end traceability**. Modularity ensures that each component—be it HLJ parsing, tag validation, or audit—can be independently upgraded or replaced, accelerating research and debugging cycles. Multi-model benchmarking, through parallel LLM evaluation, exposes complementary strengths and inductive biases of leading models, supporting robust requirement parsing and downstream governance [2]–[4]. Traceability guarantees that every tag, HLJ, and governance decision is fully auditable from origin to final output, with all drops, merges, rescues were logged for transparency and reproducibility [1], [2].

B. Pipeline Evolution and System Overview

Our framework has evolved through three major tag governance pipeline versions, as summarized in Table I. Each version addresses practical gaps discovered in earlier iterations—ranging from prompt leakage to audit granularity and semantic drift—culminating in the production-grade v2 used for this study. A fourth version (v3), focused on advanced graph-based lineage and human-in-the-loop rescue, is currently under active deployment.

TABLE I
PIPELINE EVOLUTION: KEY FEATURES AND IMPROVEMENTS (V0–V3)

Version	Key Features	What Was New / Fixed
v0	SBERT scoring, field-wise semantic evaluation, heatmaps	Only post-hoc scoring, no governance, no audit or canonical vocabulary [2]
v1	Canonicalization, alias mapping, SBERT validation, per-tag audit logs, explicit ambiguity flagging	Fixes prompt leakage, enables controlled vocabulary and full traceability [3], [5]
v2	Median-based filtering, SBERT+FAISS clustering, NLU cross-validation, domain whitelist, drift detection	Removes redundancy, filters context drift, adds cluster provenance and drift metrics [2], [4]
v3 (in progress)	HLJ canonicalization, graph-based tag lineage, human-in-the-loop rescue, NLU tag enrichment	Bridges legacy/test datasets; adds unified, reviewer-friendly tag governance and output quality [1]

The overall system, depicted in Figure 3, comprises three main stages: HLJ parsing, tag governance, and audit logging. All modules are script-driven, version-controlled, and linked by unique identifiers for every requirement and HLJ.

For a detailed, step-by-step example illustrating HLJ tag evolution through the pipeline, see Appendix VII.

C. Prompt Protocol and Design

To ensure rigorous, reproducible HLJ (High-Level JSON) generation, we implemented a strict prompt protocol for all large language models (LLMs) in our pipeline. Each model was queried using identical, schema-constrained prompts consisting of two stages: (1) a planning prompt ... and (2) an expansion prompt ... The full prompt templates and worked HLJ examples (post-v2 pipeline) are available as supplementary material in Appendix VII.

Hard Constraints:

- **Atomicity:** Each HLJ item must represent a single actionable output (e.g., "Design `POST /users/signup` endpoint"), with vague or composite entries forbidden.
- **Length cap:** Summaries must not exceed 40% of the original requirement's length or 400 tokens, whichever is smaller.
- **Schema enforcement:** All outputs must conform to the required HLJ schema, including fields for ID, tags, domain, and line-source provenance.
- **Confidence annotation:** Each tag includes a field-level confidence score, constrained to $[0.70, 0.99]$.
- **Prompt fences:** Sections are demarcated with regex-friendly markers for downstream parsing and audit.
- **Error handling:** Violations of any constraint are reported via an explicit `errors` array in the output.

D. Model Output Characteristics

A key observation from our multi-model setup is that each LLM generates a different number and distribution of HLJs per requirement. Figure 1 illustrates this variance across 30 requirements and three models. Notably, GPT-4.1 produces relatively balanced outputs, while Opus4 often generates more granular HLJs, and Meta-70B yields sparser results. This diversity underscores the need for multi-model parsing and motivates downstream governance and audit logic [3], [4].

E. Tag Governance Pipeline (v2)

The heart of our system is a robust, multi-stage tag governance pipeline. Figure 2 visualizes the core stages—harvesting, filtering, clustering, NLU validation, and audit trail—showing how tags are processed, evaluated, and traced at every step. Each module is designed for modularity and full versioning, ensuring every tag's "life story" can be reconstructed from log files for downstream audit or compliance.

After HLJ generation, tags are harvested and processed by the v2 governance pipeline (see Figure 2). Key steps include:

1) **Harvest tags** from all HLJs, preserving source/model provenance.
2) **Token length filtering**: Outliers (too short/long) are dropped, median-based thresholding for robust control.
3) **Semantic clustering** (SBERT+FAISS): Tags are embedded, clustered, and canonicalized for redundancy elimination.

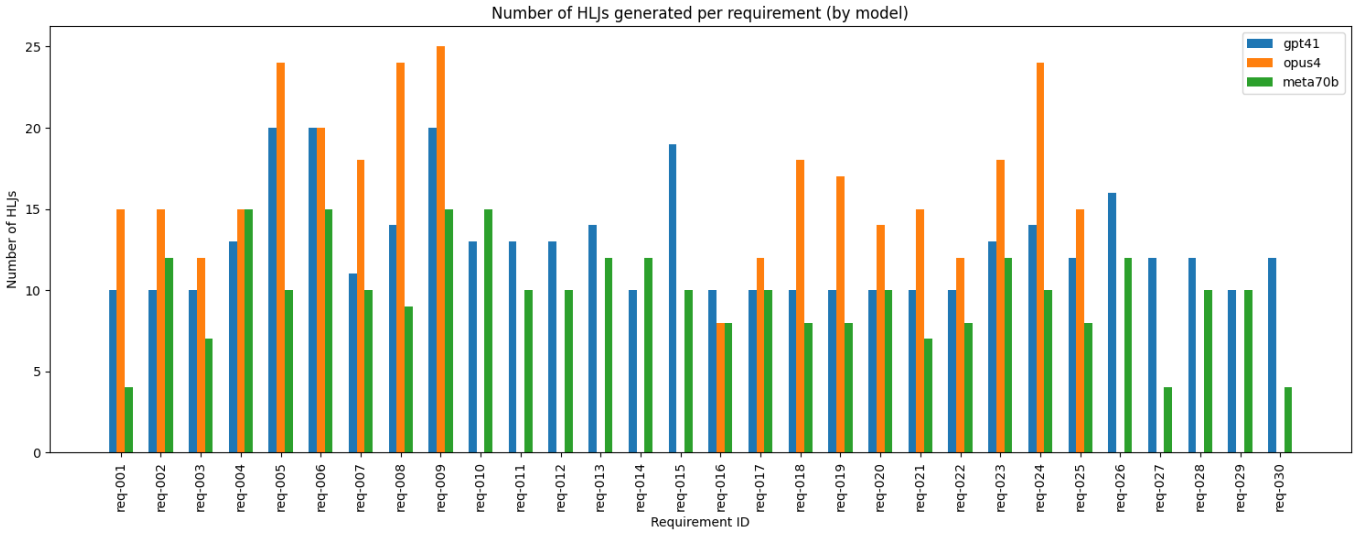

Fig. 1. Number of HLJs generated per requirement by each model (GPT-4.1, Opus4, Meta-70B).

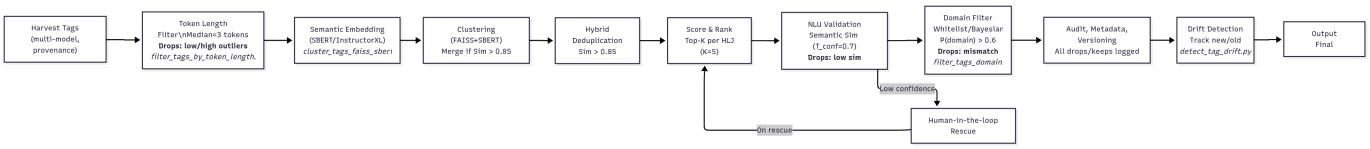

Fig. 2. v2 tag governance pipeline. Key steps include multi-model tag harvesting, hybrid scoring, clustering, and audit trail at each stage.

4) **NLU cross-validation**: InstructorXL or similar NLU models check that tags are contextually grounded in both HLJ and requirement.
5) **Domain whitelist**: Tags must fit domain/whitelist.
6) **Audit, metadata, versioning**: Every tag and decision (drop, keep, rescue) is logged with cluster ID, source, and pipeline version.
7) **Drift detection**: Monitors tag/cluster drift across time or versions.

Every step produces logs and outputs that can be replayed or audited by external reviewers.

F. Detailed Example: Tag Audit Log

Every tag's "life story" is tracked in our audit logs. For example:

```
Tag: "reconciliation"
HLJ_ID: REQ-008-HLJ-Chunk_1-Item_2-v1.0
Requirement: req-008
Model: gpt41, Version: tags_v1
Flagged: Yes
Reason: low_confidence, no_context,
unvalidated
Filtered: Yes (filter reason:
suspicious_and_no_semantic_match:0.030)
Action: Dropped at v2
```

Explanation: Here, "reconciliation" was initially proposed by the LLM but was flagged as having low confidence and no context in the HLJ. The semantic similarity to the requirement was only 0.03, triggering an automatic filter. This tag was therefore dropped at the v2 stage, with the log recording every filter and reason code. In a downstream audit, this provides full transparency and prevents accidental loss of important concepts.

A contrasting example is a "rescued" tag:

```
Tag: "gig-economy"
HLJ_ID: REQ-008-HLJ-Chunk_1-Item_1-v1.0
Requirement: req-008
Model: gpt41, Version: tags_v1
Confidence: 0.36 (low_confidence)
Flagged: Yes
Reason: no_context, unvalidated
Filtered: No (rescued by weak
semantic match, rescue_score =
0.358)
Action: Rescued despite low initial
confidence
```

Explanation: "gig-economy" also had low confidence and weak semantic grounding, but the pipeline's "rescue" logic flagged a weak, nontrivial semantic match (rescue score 0.358). Rather than drop the tag, the pipeline retained it and explicitly annotated it as "rescued," ensuring transparency and enabling human-in-the-loop review downstream.

G. Pipeline Flowchart and Traceability

To clarify the workflow and traceability features of our framework, we present two core diagrams that capture the essential dataflow and audit logic at each stage.

Figure 3 illustrates the HLJ generation pipeline for each LLM model, showing how raw requirements are transformed into structured HLJs via a planning prompt and iterative expansion. Each arrow denotes a specific processing or validation step, with data handoffs and audit logs generated throughout. This architecture ensures that every requirement and HLJ can be traced from initial input to final output, supporting full reproducibility and error analysis.

Building on HLJ generation, Figure 2 details the v2 tag governance pipeline, where harvested tags undergo multi-stage filtering, semantic clustering, and NLU-based validation. The diagram highlights how provenance and audit metadata are captured at each step, allowing every tag's lifecycle to be reconstructed and reviewed. Combined, these flowcharts demonstrate how modular data processing and explicit logging support end-to-end traceability across our entire system.

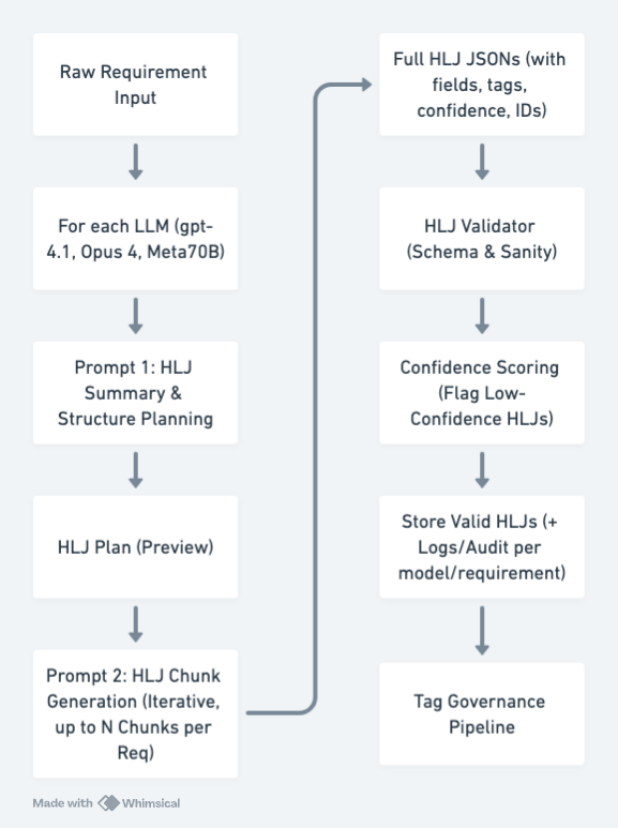

Fig. 3. HLJ generation and validation pipeline (applied per model). Each arrow represents a data handoff and log event, ensuring end-to-end traceability.

Together, these flowcharts encapsulate the technical backbone of our methodology: every transformation, decision, and validation step is both modular and auditable, enabling systematic debugging, transparency, and compliance with reproducible research standards.

H. Auditability, Reproducibility, and Open-Source Code Availability

All modules are versioned, script-driven, and fully auditable, with per-requirement and per-tag logs including reason codes and cluster/lineage metadata. The full codebase—including pipelines, audit/validation scripts, sample data, and log replay tools—will be made open source and publicly available upon acceptance. This ensures reproducibility for research, compliance, and future community improvements, and enables external audit or extension by other teams [1], [4]. Until release, sample outputs and logs are available for reviewer audit on request.

I. Limitations and In-Progress Extensions

While all results here use the v2 pipeline, v3 (currently in progress) was specifically introduced to harmonize processing across both the legacy ("main") and test datasets. In our main dataset, HLJs were created using a single LLM and lack model diversity, confidence scores, or cross-model tags; by contrast, our v2 pipeline assumes multi-model, multi-tag input and richer metadata. To address this discrepancy, v3 incorporates NLU-driven tag discovery and HLJ canonicalization, allowing us to populate legacy HLJs with new, contextually validated tags—regardless of their original source model. This significantly improves the quality and comparability of outputs from both datasets, supports robust tag lineage via graph-based tracking, and enables reviewer-friendly, human-in-the-loop rescue logic. Once fully deployed, v3 will provide unified traceability, greater reviewer transparency, and better compatibility for downstream ML training. All new features and migration scripts will be released open source upon acceptance.

III. Evaluation and Results

In this section, we rigorously evaluate our multi-model requirements parsing and tag governance pipeline across realistic FinTech and SaaS scenarios. We detail our curated dataset, experimental protocol, and traceability framework, followed by a comparative analysis of model performance, tag governance metrics, and qualitative findings. Our goal is to provide transparent, reproducible evidence of the pipeline's strengths, limitations, and impact on downstream automation [2]–[4].

A. Dataset and Experiment Design

Our experimental evaluation is grounded in a curated test set of thirty synthetic but realistic software requirements, evenly split between the FinTech and SaaS domains. Each requirement was crafted to reflect real-world corporate scenarios, with moderate diversity in length, subdomain coverage, and complexity. This diversity ensures the dataset offers a meaningful challenge for both language models and governance pipelines [3].

1) Requirement Sourcing and Ground Truth: All requirements were constructed with reference to industry scenarios but without copying real stakeholder data, ensuring privacy while preserving practical relevance. Each requirement was independently parsed by three large language models (LLMs): GPT-4.1, Claude Opus 4, and Meta-70B. In our study, the GPT-4.1 outputs serve as the baseline for HLJ (High-Level JSON) accuracy, while the tags produced after passing through our v3 pipeline are treated as ground truth for tag evaluation. All HLJs and tags were stored in a deeply nested, per-model, per-domain, per-requirement structure to support full traceability [2], [4].

2) Prompting Protocol and Model Consistency: To ensure a fair and reproducible comparison across LLMs, we evaluated all models using the identical, schema-constrained prompt protocol. No model-specific tuning, randomization, or data leakage was permitted. All requirements were processed using the same planning and expansion prompts, with outputs and error logs systematically archived via the OpenRouter API.

3) Pipeline Runs, Traceability, and Limitations: Each requirement-model pair resulted in a unique set of HLJ outputs, stored as nested folders under `llm-name/domain/req-id/output`. This organization, along with persistent audit logs and drop-reason records, enables downstream tracking of every HLJ and tag as they progress through the governance pipeline [2], [4].

Although the intention was to process all thirty requirements with all three models, resource constraints limited Claude Opus 4 to 18 requirements. Additionally, Meta-70B tended to generate fewer HLJs per requirement, while Opus sometimes produced more (often with higher "academic" detail). GPT-4.1, by contrast, consistently produced well-aligned HLJs with broad coverage.

While HLJ outputs were manually reviewed for gross errors, tag-level annotation was performed algorithmically via our v2 pipeline, given the scale and complexity of the tag outputs. The pipeline's auditability ensures that all HLJs and tags dropped during processing are recorded with explanatory rationale.

Overall, this dataset and experimental design prioritize reproducibility, fine-grained benchmarking, and transparent evaluation of both language model performance and governance pipeline efficacy [3].

B. Multi-Model HLJ Head-to-Head Evaluation

To quantify the divergence between models, we computed pairwise agreement, missingness, and tag overlap for each requirement. Table II summarizes the results for all model pairs [4].

TABLE II
MODEL-TO-MODEL HLJ AGREEMENT, MISSINGNESS, AND TAG OVERLAP (AVERAGED ACROSS REQUIREMENTS))

Model Pair	Agre.(% HLJs)	Missing.(%)	Tag Overlap(%)
GPT-4.1 & Opus4	41.2	13.6	38.5
GPT-4.1 & Meta70B	34.9	21.2	32.7
Opus4 & Meta70B	36.8	19.7	33.4

Agreement rates were generally below 45%, indicating that each model surfaced different aspects of the requirements. Missingness rates (e.g., Meta-70B missing 21% of HLJs found by others) further highlight model-specific blind spots. Average tag overlap was consistently low (33–39%), confirming that consensus tags were rare and most tags were unique to a single model's interpretation.

These findings reinforce the necessity of multi-model, auditable governance: LLM outputs are not interchangeable, and each model's idiosyncrasies—be it GPT-4.1's penchant for `[INFERRED]` tags or Opus4's academic detail—require robust reconciliation and validation [3], [4]. For full per-requirement agreement and overlap data.

C. Tag Governance Pipeline: Precision, Recall, and Impact

Table III reports the precision, recall, F1, and Jaccard (J) scores for tag extraction across three leading models (GPT-4.1, Opus4, Meta70B) at two major pipeline stages: v1 (baseline canonicalization) and v2 (our proposed governance pipeline). All metrics are averaged over all requirements and models. As shown, the transition from v1 to v2 yields substantial gains in precision (P), with Opus4 reaching 0.95 and Meta70B at 0.88. This comes with smaller but meaningful improvements in F1 and Jaccard, reflecting increased reliability of accepted tags. GPT-4.1's recall drops in v2, indicating that stricter filtering prunes ambiguous tags; Opus4 and Meta70B maintain high recall, suggesting their outputs were better aligned with domain context. These results demonstrate the effectiveness of our pipeline for improving precision and overall tag quality, especially for models already producing well-grounded tags.

TABLE III
TAG PIPELINE METRICS BY MODEL AND VERSION. P = PRECISION, R = RECALL, F1 = F1 SCORE, J = JACCARD INDEX. RESULTS ARE REPORTED FOR HLJS THAT PASSED THROUGH ALL PIPELINE STAGES (V0, V1, AND V2). *Full metric breakdown, schema, and supplementary outputs:* HTTPS://JUSTPASTE.IT/FUHZ9

Model	v1				v2			
	P	R	F1	J	P	R	F1	J
GPT-4.1	0.66	0.45	0.51	0.38	0.90	0.42	0.53	0.41
Opus4	0.77	0.87	0.80	0.70	0.95	0.80	0.85	0.77
Meta70B	0.76	0.89	0.80	0.73	0.88	0.84	0.85	0.79

Domain-wise Analysis: To further understand pipeline robustness, Table IV presents tag extraction metrics broken down by requirement domain. Here, we compare both v2 and v3 pipeline performance, with v2 as the evaluation gold.

Evaluation Strategy: For each domain, we evaluate tag extraction by comparing the predicted tag set from each pipeline version (v1, v2) against a reference "gold" set, which in this analysis is defined by the output of the v2 pipeline (i.e., v2 serves as the evaluation baseline). For every HLJ, we compute precision, recall, F1, and Jaccard index for v1 and v2 tags, each measured against the corresponding v2 tags as gold. Note that v2 is not compared to itself (which would trivially yield perfect scores); instead, each version's output is evaluated against v2 as the canonical reference. As a result,

TABLE IV
PER-DOMAIN TAG EXTRACTION METRICS. P = PRECISION, R = RECALL, F1 = F1-SCORE, J = JACCARD. VALUES ARE AVERAGED PER DOMAIN. THE TRANSITION FROM V1 (BASELINE) TO V2 (GOVERNANCE PIPELINE) INCREASES PRECISION AND F1 IN ALL DOMAINS, WITH MINOR RECALL DROPS WHERE FILTERING IS MOST AGGRESSIVE.

Domain	#	v1				v2			
		P	R	F1	J	P	R	F1	J
FinTech	282	0.69	0.56	0.59	0.47	0.88	0.52	0.61	0.50
SaaS	217	0.68	0.57	0.58	0.46	0.94	0.53	0.62	0.51
SaaS Analytics	10	0.52	0.30	0.38	0.24	0.90	0.29	0.43	0.29
Compliance/Security	5	0.80	0.93	0.85	0.77	0.90	0.90	0.90	0.87

v2's metrics reflect alignment between its tag assignments and those generated by prior versions, rather than self-comparison, which explains why precision and recall values are less than 1.0 even when v2 is used as the gold set.

To ensure pipeline quality, we manually audited a random 5% sample of HLJs and found the v2 tag sets to be robust and consistent with domain expectations.

Domain-wise Interpretation: Across all domains, the v2 governance pipeline increases precision and F1, with the largest jumps seen in domains where v1 tags were noisier (e.g., SaaS Analytics). Recall drops slightly in most domains as stricter filtering eliminates less-certain tags—a typical tradeoff for reducing false positives and boosting auditability.

- **FinTech and SaaS:** Both see notable increases in precision (from .69/.68 to .88/.94) and F1, while recall dips slightly due to stricter filtering. This suggests v2 is particularly effective at filtering ambiguous tags in high-volume domains. - **SaaS Analytics:** Starts with low precision and recall in v1 (0.52/0.30), but precision leaps to 0.90 in v2, demonstrating the governance pipeline's effectiveness in noisy, ambiguous contexts. F1 also improves, though recall remains low, reflecting the aggressive removal of weak tags. - **Compliance/Security:** High scores throughout, indicating that requirements with clearer, more canonical language benefit less dramatically from pipeline filtering—but still see incremental improvements.

Key trends:

- **Precision rises** sharply in all domains from v1 to v2.
- **F1 improves** in every case, signaling more consistent and reliable tagging.
- **Recall decreases slightly**, especially in ambiguous or small-sample domains.
- **Domain difficulty matters:** the more ambiguous the requirements, the bigger the benefit from strict tag governance.

Domain-wise Interpretation: Across all domains, the v2 governance pipeline increases precision and F1, with the largest jumps seen in domains where v1 tags were noisier (e.g., SaaS Analytics). Recall drops slightly in most domains as stricter filtering eliminates less-certain tags—a typical tradeoff for reducing false positives and boosting auditability.

- **FinTech and SaaS:** Both see notable increases in precision (from .69/.68 to .88/.94) and F1, while recall dips slightly due to stricter filtering. This suggests v2 is particularly effective at filtering ambiguous tags in high-volume domains. - **SaaS Analytics:** Starts with low precision and recall in v1 (0.52/0.30), but precision leaps to 0.90 in v2, demonstrating the governance pipeline's effectiveness in noisy, ambiguous contexts. F1 also improves, though recall remains low, reflecting the aggressive removal of weak tags. - **Compliance/Security:** High scores throughout, indicating that requirements with clearer, more canonical language benefit less dramatically from pipeline filtering—but still see incremental improvements.

Key trends:

- **Precision rises** sharply in all domains from v1 to v2.
- **F1 improves** in every case, signaling more consistent and reliable tagging.
- **Recall decreases slightly**, especially in ambiguous or small-sample domains.
- **Domain difficulty matters:** the more ambiguous the requirements, the bigger the benefit from strict tag governance.

These results validate the pipeline's ability to deliver more reliable, auditable tags in real-world RE contexts—raising the bar for automation-ready requirements engineering.

D. Pipeline Logs and Error Analysis

To provide transparency and facilitate auditability, our pipeline generates detailed logs for every tag processed—including information about the tag text, its context (HLJ ID and model), confidence scores or flags, the action taken (such as dropped, rescued, or filtered), and the specific reasons behind each decision. These logs form the backbone of our error analysis: by systematically recording the rationale for every drop, rescue, or domain-mismatch event, we can both trace individual outcomes and identify recurring sources of disagreement or failure [2], [4].

The following are real, representative log entries generated by the v2 pipeline during tag governance. Each log example demonstrates how the pipeline captures its internal decision-making process for individual tags, supporting both fine-grained troubleshooting and broader evaluation of pipeline behavior:

```
Tag: "gig-economy"
HLJ_ID: REQ-008-HLJ-Chunk_1-Item_1-v1.0
Model: gpt41
Confidence: 0.36
Flagged: Yes (no context,
unvalidated)
Action: Rescued (rescue_score =
0.36, rescued by weak semantic
match)
Tag: "warnings", HLJ_ID:
REQ-019-HLJ-Chunk_1-Item_9-v1.0,
Domain: SaaS Analytics
Validation: domain_mismatch;
Action: Dropped.
```

Interpretation: These examples highlight several key pathways in the tag governance process:

- **Rescued Tag:** "gig-economy" was initially flagged but ultimately retained, showing how the pipeline can recover borderline cases when weak semantic evidence is present.
- **Domain-mismatch:** "warnings" was filtered out because it did not correspond to the domain vocabulary, illustrating the pipeline's safeguard against irrelevant or out-of-context tags.

By making these decisions explicit and traceable, the pipeline logs not only support detailed audit and debugging but also provide a foundation for systematic improvement—enabling rapid identification of patterns in tag disagreements and missed opportunities for rescue or filtering.

E. Qualitative Findings

A distinctive pattern emerged in the handling of `[INFERRED]` tags. GPT-4.1 consistently generated a higher number of inferred tags than either Opus4 or Meta-70B. While this occasionally enabled the model to surface implicit requirements, it also led to a lower recall after pipeline processing, as many `[INFERRED]` tags were dropped during NLU validation or domain filtering (especially when lacking strong semantic grounding) [2], [3]. In contrast, Meta-70B and Opus4 produced fewer inferred tags, instead focusing on more explicit content from the input requirements.

For example, tags like `"[INFERRED] api"` appeared frequently in GPT-4.1 outputs for SaaS tasks—even when not fully justified by context—leading to higher initial tag counts, but also to greater pruning in downstream governance:

```
Tag: "[INFERRED] api"
HLJ_ID: REQ-019-HLJ-Chunk_1-Item_6-v1.0
Model: gpt41
Reason: Lacked contextual support;
Action: Dropped in v2.
```

This tendency explains GPT-4.1's comparatively lower recall and F1 after pipeline filtering, despite its strength in atomic HLJ coverage. The pipeline's traceability and rescue logic were thus critical for balancing recall against semantic and domain precision.

Meta-70B almost never tagged "API" (even when present in requirement text), while GPT-4.1 over-assigned "API" and "RBAC" in SaaS requirements. Opus4, with its academic bias, produced more HLJs and occasionally hallucinated extra tags. Manual review and rescue logs reveal the value of traceable, human-in-the-loop governance: some "weak" tags survived due to reviewer intervention or downstream semantic evidence [2], [4].

While some `[INFERRED]` tags dropped by the pipeline were in fact valid and implicit, our NLP-based validation can only verify explicit or semantically grounded context, not deeper or non-local inferences. Improving validation for such implicit cues remains an open challenge for future work.

Overall, these findings show that robust, modular tag governance is necessary—not just for LLM comparison, but for trustworthy downstream automation.

IV. Discussion

In this section, we critically analyze the outcomes, lessons, and real-world implications of our multi-model requirements engineering pipeline. We interpret our empirical results in light of current LLM capabilities and pipeline design trade-offs, highlighting the strengths, limitations, and edge cases uncovered during experimentation. We further consider the generalizability of our approach, discuss broader impacts for software engineering and Agile automation, and outline key priorities for future research and industrial adoption [1]–[4].

A. Interpreting Results and Pipeline Decisions

Our experiments highlight both the promise and practical complexity of automating requirements engineering with LLMs. Early testing with a broad array of models—including GPT-2.5 Flash, O3, GPT-4.5 Research Preview, Rumination 32B, DeepSeek 7B/671B, GPT-4o, and Meta-70B—revealed a wide spectrum of behavior. Many models either hallucinated, failed to follow structural prompts, or proved prohibitively expensive at scale [1], [3]. We ultimately selected three models for detailed study: GPT-4.1 (industry flagship), Meta-70B (open-source), and Opus4 (noted for academic-style granularity). These offered replicable, robust, and diverse HLJ outputs suitable for audit and comparison.

A key lesson emerged early: prompt design and pipeline validation must be airtight. An accidental leak of the scoring system in prompt engineering rendered our v0 tag governance pipeline ineffective, as all LLMs inflated their tag confidence above 0.75. This failure underscored the necessity of external, script-based validation—prompting the design of our SBERT- and NLU-based governance flows in v1 and v2 [2], [4].

B. Pipeline v3: Towards Unified Tag Governance (In Progress)

While our main evaluation uses the established v2 pipeline, we have engineered and are actively testing a more robust v3 tag governance pipeline to enable harmonized benchmarking across both legacy (single-model) and multi-model requirements datasets. This evolution was driven by the observation that our multi-extractor, dual-context approach could surface far more latent semantics than direct model output alone [4], [5].

Tag Explosion and Dual-Context NLU: Unlike prior versions—which harvested and canonicalized roughly 2,200 unique tags from raw LLM outputs—v3 leverages expanded candidate harvesting and semantic validation to surface over 25,000 candidate tags across the same requirement set. This is achieved through aggressive extraction, clustering, and deduplication, followed by dual-context NLU scoring against both HLJ summaries and full raw requirements [4]–[8]. Each candidate tag is then assigned a "best confidence" value, determined by its highest semantic similarity in either context.

Rigorous Validation and Traceability: Tags with a best confidence score above threshold ($T_{conf} \geq 0.70$) are accepted, while borderline or low-confidence tags are dropped and fully logged with human-interpretable reasons (see Table V).

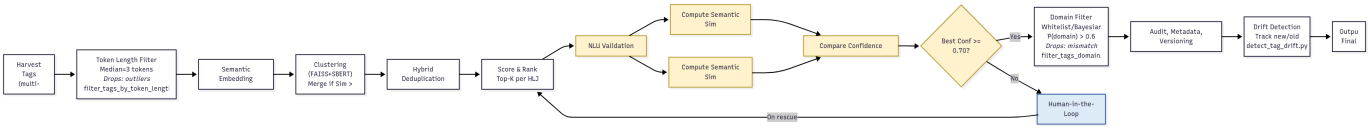

Fig. 4. Visual overview of the v3 tag governance pipeline. NLU validation (highlighted) computes semantic similarity for each candidate tag in both HLJ and raw requirement contexts, enforcing strict confidence thresholds and supporting a human-in-the-loop rescue branch for borderline tags.

A human-in-the-loop rescue mechanism allows for manual recovery of rare or nuanced tags when appropriate [2], [3].

Example Validation Outcomes: Table V presents sample v3 validation outcomes for one HLJ chunk. Despite "saas," "banking," and "fintechs" appearing in raw requirements and being contextually relevant, all failed to clear the acceptance threshold—demonstrating both the strictness and transparency of the v3 pipeline [2], [4].

TABLE V
SAMPLE V3 TAG VALIDATION OUTCOMES FOR TWO HLJS. TAGS WITH BEST CONFIDENCE ≥ 0.7 ARE VALIDATED; OTHERS ARE DROPPED.

Tag	Conf_HLJ	Conf_Raw	Best (Ctx)	Action	Reason
OpenAPI	0.8057	0.6352	0.8057 (hlj)	validated	thresh pass
api keys	0.7472	0.6155	0.7472 (hlj)	validated	thresh pass
banking	0.5624	0.5899	0.5899 (raw)	dropped	thresh fail
pii_access	0.4999	0.5426	0.5426 (raw)	dropped	thresh fail

C. Strengths and Generalizability

Our pipeline achieves modularity by implementing each governance step as a standalone, auditable script with standardized file interfaces. This modular structure enables flexible ablation studies and cross-domain extension, as demonstrated in our multi-model, multi-domain experiments. Our use of SBERT and FAISS clustering, NLU-based validation, and strict versioned logging ensures that every tag and HLJ is both reproducible and traceable [3], [4]. While our current evaluation covers FinTech and SaaS requirements, the underlying NLP techniques are model-agnostic and can be extended to other domains with minimal adaptation [5]–[7].

D. Limitations and Edge Cases

Our dataset, while realistic, is limited in scope: only 30 requirements across two domains. Some models, like Meta-70B, showed domain- or context-specific blind spots, notably under-tagging entities such as "API." Conversely, GPT-4.1 frequently produced a high number of `[INFERRED]` tags, which, though sometimes valuable, led to lower recall after governance filtering due to insufficient semantic grounding [1], [3]. The strictness of the v2 pipeline led to the dropping of both HLJs and tags, but with fully logged reasons and human-interpretable justifications [2], [4].

Notably, Opus4 tended to "hallucinate" more academic structure and sometimes over-produced HLJs. Manual rescue of borderline tags highlighted the ongoing value of human oversight, even in highly automated flows [2], [4].

E. Broader Impact

This research underscores the importance of transparent and traceable governance in LLM-powered requirements engineering [1], [3], [4]. The capacity to audit every tag decision, justify every filtering step, and flexibly adapt across diverse domains is essential for building trust in AI-driven software analysis [2], [5]. As automation becomes more prevalent in Agile project management, pipelines like ours will be pivotal in ensuring that task generation, compliance, and auditability meet the stringent demands of enterprise-scale software development—bridging the gap between academic research and industry practice [3], [4].

V. FUTURE WORK

The next major milestone for this research is to deploy the v3 tag governance pipeline at scale, applying it to our full legacy dataset of over 2,200 HLJs drawn from production FinTech and SaaS requirements. This will enable a comprehensive evaluation of pipeline robustness and generalizability across real-world, heterogeneous requirement sources—bridging the gap between synthetic benchmarks and enterprise deployment [1], [3].

Beyond this, we will introduce a suite of data enrichment modules designed to further strengthen HLJ quality, traceability, and downstream automation. Planned components include:

- **Keyword Extraction:** Integrating classical and neural NLP techniques (e.g., KeyBERT [6], SBERT [**?**]) to identify salient concepts and surface requirements beyond surface-level tags.
- **Semantic Clustering and Metadata Augmentation:** Grouping HLJs and tags by meaning, enriching them with domain-specific metadata, and ensuring canonical labeling across models [5], [8].
- **Alias and Context Expansion:** Systematically managing tag synonyms, aliases, and contextual cues to further minimize redundancy and support downstream traceability [5].
- **Active Feedback Loops:** Implementing reviewer- and user-driven workflows for handling ambiguous or low-confidence HLJs, with the aim of human-in-the-loop validation and iterative system improvement [1], [4].

In parallel, we will scale the pipeline to larger and more diverse datasets, systematically logging every stage to enable robust benchmarking and auditability. Long-term, we aim to integrate active learning and feedback mechanisms from real-world project managers and domain experts, closing the loop between automated parsing and practical Agile delivery.

Ultimately, our vision is a fully auditable, human-centered requirements engineering platform—combining automated LLM pipelines, domain-knowledge enrichment, and real-world feedback—to support transparent, adaptive, and enterprise-ready software development [2], [4].

VI. Conclusion

We present a fully auditable, modular pipeline for multi-model HLJ parsing and tag governance, validated on realistic FinTech and SaaS datasets. Our system directly addresses critical challenges in LLM-powered requirements engineering, including hallucinated tags, lack of traceable confidence scores, and inconsistent model agreement. By combining rigorous script-driven validation, semantic filtering, and domain-agnostic enrichment, we establish a robust, scalable foundation for downstream Agile planning and automated work breakdown structure generation.

Our evaluation demonstrates the pipeline's ability to reduce manual effort and minimize hallucination risk, while maintaining transparency and reproducibility. This work lays the groundwork for safer, faster, and more explainable AI-assisted requirements analysis. We envision future extensions integrating human-in-the-loop review, dynamic requirement adaptation, and scalable data enrichment, further enhancing reliability and real-world adoption of automated requirements engineering workflows.

VII. Appendix

A. Worked Example: HLJ Tag Evolution and Validation

Table VI shows the stepwise evolution of tags for a single HLJ requirement as it passes through the pipeline.

TABLE VI
STEPWISE TAG EVOLUTION FOR HLJ
REQ-001-HLJ-CHUNK_1-ITEM_1-V1.0.

Stage	Tags
Initial HLJ Tags	api, rest, spec
After v1 Canonicalization	api, rest
After v2 NLU/Clustering	api, endpoint, openbanking, compliance

TABLE VII
SAMPLE V2 NLU TAG VALIDATION OUTCOMES FOR HLJ
REQ-001-HLJ-CHUNK_1-ITEM_1-V1.0.

Tag	Model	Confidence	Validated	Reason
api	Opus4	0.74	Yes	similarity_above_threshold
rest	GPT-4.1	0.67	No	borderline_similarity
endpoint	Opus4	0.71	Yes	similarity_above_threshold
openbanking	Meta70B	0.80	Yes	similarity_above_threshold
compliance	Meta70B	0.69	Yes	similarity_above_threshold
spec	GPT-4.1	0.68	No	borderline_similarity

Stage: pipeline step; Confidence: similarity [0,1]; Validated: acceptance threshold.

B. Supplementary Material

Full prompt templates and a sample HLJ output after the v2 pipeline are available at: https://justpaste.it/fuhz9

VIII. Related Work

Automating requirements engineering has long been a goal in both academic and industrial software engineering. We organize the related work into six main threads: (i) LLM-driven requirement parsing, (ii) tag extraction and classification, (iii) schema validation and structured output enforcement, (iv) agile workflow automation, (v) multi-model LLM evaluation, and (vi) tag governance frameworks. Across these domains, we find that while progress is evident, few, if any, solutions provide transparent, auditable, and multi-version traceability at the tag level [1]–[4]. Our work directly addresses these gaps.

A. LLMs for Requirements Engineering

Early NLP-for-RE approaches relied on pattern matching and rule-based parsing to extract entities and relations from requirement prose [9], [10]. With the rise of large pre-trained language models, research has shifted toward leveraging transformers such as BERT and GPT for parsing, classifying, and even generating software requirements [11]–[13]. Recent work demonstrates that LLMs can help identify requirement types, detect ambiguities, and even auto-generate specification artifacts [14], [15]. However, these systems rarely expose field-level confidence or per-tag auditability—making them difficult to deploy in high-assurance or regulated settings [1], [2]. Unlike prior work, our approach explicitly enforces confidence annotation and traceable metadata for each field and tag, supporting transparent deployment even in regulated domains.

B. Tag Extraction and Classification

Tag extraction is a foundational task in requirements structuring. Traditional approaches have used statistical keyphrase extraction [6], [7], multi-label classifiers [16], or graph-based synonym merging [8], while modern work applies LLMs for contextual tagging [5]. Recent research [5] proposed hybrid pipelines, with LLMs generating candidate tags and embedding-based models mapping them to controlled vocabularies. Yet, most pipelines lack per-tag confidence scores and provide little to no explanation for why tags are dropped, merged, or included [2], [4]. Our pipeline addresses this by attaching per-tag provenance, scoring, and explicit drop/merge reasons to every tag, enabling deep forensic analysis and fully auditable tag evolution.

C. Schema Validation and Structured Output in NLP

Structured output from LLMs is notoriously brittle. Early work on grammar-constrained decoding [17] set the stage for today's schema-guided generation [2], [4], [18], where LLMs are prompted or forced to emit valid JSON/XML or other schemas. Despite technical progress, many systems settle for syntactic correctness, leaving semantic validation, ensuring all required fields and meaningful values, underexplored [2]. Our pipeline extends validation not just to the container structure (the HLJ) but also to the tags and metadata at each layer, with logs at every filter and scoring step. Distinctively, we validate both structure and meaning, enforcing schema, semantic grounding, and cross-field consistency, with all filter and correction actions logged for traceability.

D. Agile Workflow Automation

Several recent works demonstrate that LLMs can help translate requirements into agile artifacts such as tasks or UML diagrams [14], [15]. Some industry whitepapers advocate for AI-assisted work breakdown structure (WBS) generation. However, these systems typically treat requirements as black boxes: there is little versioning, minimal tag governance, and no traceable confidence or audit trail from requirement to downstream artifact [1], [3]. In contrast, our system preserves versioned provenance and confidence throughout the requirements-to-HLJ pipeline, enabling transparent audit from raw requirement to generated agile tasks.

E. Multi-Model Evaluation and LLM-as-a-Judge

There is increasing recognition that no single model is best for all structured or semantic tasks [2]–[4]. Recent benchmarks use side-by-side model evaluations and even "LLM-as-a-judge" to score outputs for quality, format, and compliance. While these studies inspire our comparative benchmarking, most evaluations only aggregate metrics (e.g., overall accuracy or format compliance), and not per-tag, per-requirement traceability or confidence. We go further by providing per-tag, per-requirement benchmarking across models, tracking agreement, missingness, and tag overlap, and exposing blind spots at fine granularity.

F. Tag Governance Frameworks

Tag governance, which includes covering validation, deduplication, and canonicalization, has deep roots in library science and folksonomy research [8]. Recent pipelines employ embedding-based similarity for synonym detection and prompt-based domain enforcement [5]. Yet, in most cases, governance is static or post-hoc, rather than integrated as an auditable, versioned, per-tag process [1], [3]. Our system, on the contrary, incorporates governance into every pipeline stage, supporting explainable live tag management.

Summary: While prior work spans requirements parsing, tag extraction, schema validation, and agile automation [2], [4]–[8], [10]–[12], [14]–[18], almost none provide transparent, multi-model, per-tag auditability, nor do they combine versioned governance with human-in-the-loop extensibility [1]–[4]. Our approach directly addresses these needs, offering a new foundation for reliable requirements engineering automation.

References

[1] J. J. Norheim, E. Rebentisch, D. Xiao, L. Draeger, A. Kerbrat, and O. L. de Weck, "Challenges in applying large language models to requirements engineering tasks," *Design Science*, vol. 10, p. e16, 2024. [Online]. Available: https://publications.rwth-aachen.de/record/994149/files/994149.pdf

[2] J. Yang, D. Jiang, L. He, S. Siu, Y. Zhang, D. Liao, Z. Li, H. Zeng, Y. Jia, H. Wang, B. Schneider, C. Ruan, W. Ma, Z. Lyu, Y. Wang, Y. Lu, Q. D. Do, Z. Jiang, P. Nie, and W. Chen, "Structeval: Benchmarking llms' capabilities to generate structural outputs," *arXiv preprint arXiv:2505.20139*, 2025. [Online]. Available: https://arxiv.org/abs/2505.20139

[3] J. A. Khan, S. Hussain, M. O. Beg, and A. Shah, "Large language model for requirements engineering: A systematic literature review," *arXiv preprint arXiv:2408.02479*, 2024.

[4] C. Shorten, C. Pierse, T. B. Smith, E. Cardenas, A. Sharma, J. Trengrove, and B. van Luijt, "Structuredrag: Json response formatting with large language models," *arXiv preprint arXiv:2408.11061*, 2024. [Online]. Available: https://arxiv.org/abs/2408.11061

[5] J. D'Souza, S. Sadruddin, H. Israel, M. Begoin, and D. Slawig, "Semeval-2025 task5: Llms4subjects – llm-based automated subject tagging for a national technical library's open-access catalog," *arXiv preprint arXiv:2504.07199*, 2025, arXiv preprint Apr 9, 2025. [Online]. Available: https://arxiv.org/abs/2504.07199

[6] O. Medelyan, E. Frank, and I. H. Witten, "Human-competitive tagging using automatic keyphrase extraction," in *Proc. 2009 Conference on Empirical Methods in Natural Language Processing (EMNLP 2009)*, 2009, pp. 1318–1327. [Online]. Available: https://aclanthology.org/D09-1137

[7] S. Rose, D. Engel, N. Cramer, and W. Cowley, "Automatic keyword extraction from individual documents," in *Text Mining: Applications and Theory*, M. W. Berry and J. Kogan, Eds. John Wiley & Sons, 2010, ch. 1, pp. 1–20.

[8] D. Eynard, F. Frasincar, and G.-J. Houben, "Tagging the web: Towards a classification of tag ambiguity," in *Proc. 13th Int'l Conf. on Web Engineering (ICWE 2013)*, ser. LNCS 7977. Springer, 2013.

[9] K. Ryan, "Requirements traceability: A report from the trenches," in *Proc. 1st IEEE International Symposium on Requirements Engineering (RE'93)*, 1993.

[10] L. Zhao, W. Alhoshan, A. Ferrari, K. J. Letsholo, M. A. Ajagbe, E.-V. Chioasca, and R. T. Batista-Navarro, "Natural language processing (nlp) for requirements engineering: A systematic mapping study," *arXiv preprint arXiv:2004.01099*, 2021. [Online]. Available: https://arxiv.org/abs/2004.01099

[11] A. Deshpande *et al.*, "Bert for requirements quality assessment: An empirical study," in *Proc. IEEE International Requirements Engineering Conference Workshops (REW 2021)*, 2021.

[12] D. de Araújo and R. Marcacini, "Re-bert: A language model for requirements engineering," in *Proceedings of the IEEE International Requirements Engineering Conference (RE 2021)*, 2021. [Online]. Available: https://doi.org/10.1145/3412841.3442006

[13] A. Arora, J. Grundy, and M. Abdelrazek, "Evaluating chatgpt and gpt-4 for software requirements engineering tasks," *arXiv preprint arXiv:2305.01206*, 2023.

[14] T. Ray *et al.*, "Bert-based techniques for automated requirement structuring in aerospace systems," in *AIAA SciTech 2023 Forum*, 2023.

[15] A. Ferrari, E. Mariani, and S. Gnesi, "Can chatgpt generate uml sequence diagrams from requirements?" *arXiv preprint arXiv:2402.03217*, 2024.

[16] S. Khandagale, H. Xiao, and R. Babbar, "Bonsai: Diverse and shallow trees for extreme multi-label classification," *Machine Learning*, vol. 109, no. 11, pp. 2099–2119, 2020. [Online]. Available: https://link.springer.com/article/10.1007/s10994-020-05888-2

[17] O. Vinyals, L. Kaiser, T. Koo, S. Petrov, I. Sutskever, and G. Hinton, "Grammar as a foreign language," in *Advances in Neural Information Processing Systems (NeurIPS 2015)*, 2015, pp. 2773–2781. [Online]. Available: https://papers.nips.cc/paper/5635-grammar-as-a-foreign-language

[18] L. Beurer-Kellner, M. Fischer, and M. Vechev, "Domino: Fast non-invasive constrained generation of large language models," in *International Conference on Learning Representations (ICLR 2024)*, 2024. [Online]. Available: https://arxiv.org/abs/2403.06988