Human-Validated Reverse Engineering as Semantic Conditioning for Media Generation from Technical Artifacts

A Paired Comparison of Direct Code Prompting and Requirements-Mediated Prompting Across Proprietary Software Systems

Matthias Harter
Faculty of Engineering
Hochschule RheinMain - University of Applied Sciences
Rüsselsheim, Germany
e-mail: matthias.harter@hs-rm.de

Rafael Zajonz
Market Intelligence and Quantitative Analysis
Deutsche Bundesbank
Frankfurt am Main, Germany
e-mail: rafael.zajonz@bundesbank.de

***Abstract*—This paper presents a pilot study on whether human-validated requirements extracted from technical artifacts improve stakeholder-facing media generation when compared with direct code-to-media prompting. A six-phase pipeline preprocesses repository contents, performs reverse engineering on source code, subjects the resulting requirement candidates to manual validation, derives audience profiles from role-goal-centered knowledge assessment, generates media in two experimental conditions, and evaluates the outputs with claim-based, semantic, coverage, profile-fit, and readability metrics. The corpus comprises four proprietary systems that were not publicly available and therefore reduce the risk of training-set contamination: Bundesbank-Bot (BUBA-Bot) , Flight Data Recorder, Legal Chatbot, and MSP430-Simulator (MSP-Sim). The set spans Python systems, embedded C firmware for the MSP430, and a simulator with substantial auto-generated C code produced from a graphical model. The central comparison contrasts direct code prompting with requirements-mediated prompting for report and audio artifacts. Across the main metrics, the requirements-mediated condition reduces hallucination rates, increases problem-space focus and requirement coverage, and lowers technical bleed. The design treats reverse engineering as a semantic control layer that supports human validation before downstream communication to non-technical stakeholders.**

Keywords*—*reverse engineering; requirements engineering; technical communication; audience-adaptive media generation; software artifact analysis; hallucination evaluation.

I. INTRODUCTION

Software engineering is entering a phase in which implementation is no longer the only scarce resource. Coding agents can generate files, refactor modules, execute multi-step debugging workflows, and return control to a human reviewer only at selected checkpoints. Gartner now expects Artificial Intelligence (AI) code assistants to be used by 90% of enterprise software engineers by 2028 and explicitly frames the developer role as shifting from implementation toward orchestration, problem solving, and system design. In parallel, low-code platforms have become routine in enterprise development, and the recent notion of *vibe coding* captures a more conversational mode of software production in which the human increasingly specifies intent while the system generates, debugs, and revises the concrete implementation [1] [2] [3] [4].

The economic consequence is direct. Time-to-market and non-recurring engineering cost are pushed by a different optimization frontier once an agent can assemble a first working implementation, run tests, inspect failures, and revise the code in tight loops. Manual implementation ceases to be the natural center of the process. The pressure on human engineers moves upward, toward architectural decomposition, interface definition, compliance constraints, and acceptance criteria, and outward, toward the validation of business intent and stakeholder communication. No-code and low-code environments were an earlier expression of this direction; agentic coding systems extend it into domains that previously required fully manual programming [2] [3] [5] [4].

There is historical precedent for such abstraction shifts. Compiler construction displaced hand-written machine code. Model-based engineering and qualified code generators displaced parts of low-level implementation review in safety-critical settings. Ansys reports that the SCADE Display C code generator (KCG) is qualified up to DO-178C/DO-330 Tool Qualification Level 1 (TQL-1) and software safety levels up to ISO 26262 Automotive Safety Integrity Level D (ASIL D). In such settings, engineers do not inspect generated low-level detail line by line; trust is relocated to the abstraction, the generator, and the verification regime. Large Language Model (LLM) based generation changes the picture, however. Its outputs are not derived by a semantics-preserving translation in the classical compiler sense. They are probabilistic, prompt-sensitive, and capable of producing plausible but semantically misaligned implementations. This is why current industrial practice already surrounds generation with review and validation agents, such as CodeRabbit, GitHub Copilot code review, and Greptile, all of which operate on pull requests rather than replacing validation altogether [6] [7] [8] [9].

This paper starts from that asymmetry. If agentic systems increasingly dominate the solution space, human control cannot remain anchored in line-by-line implementation review. It has

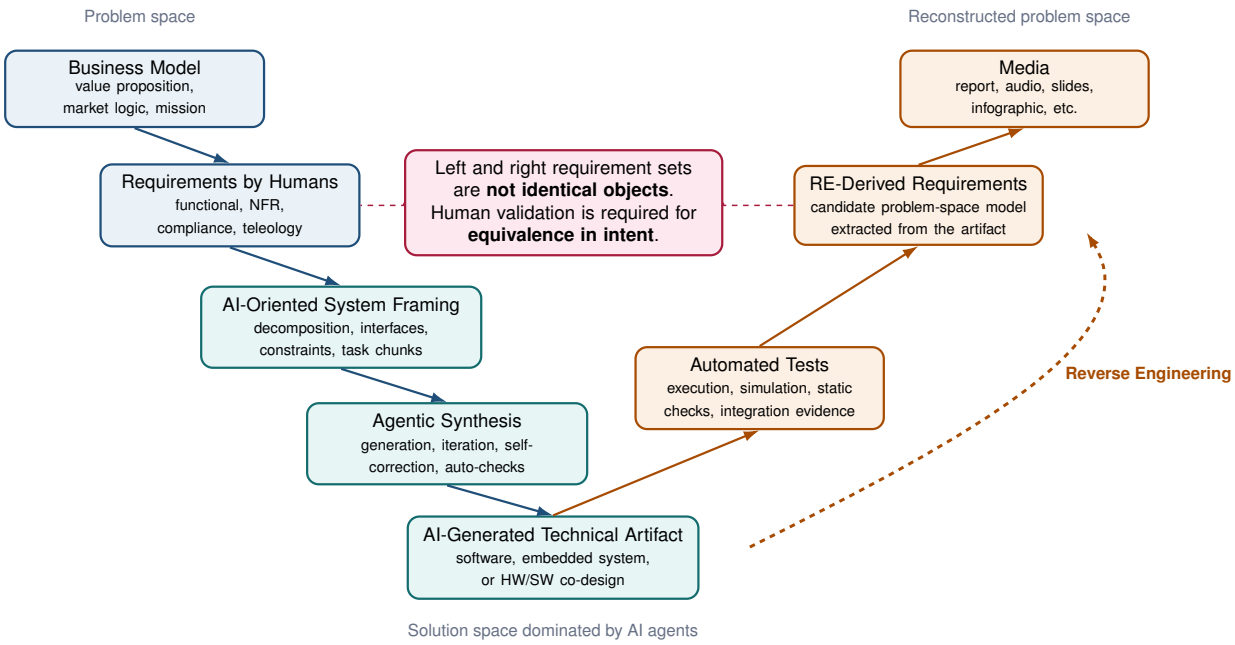

Figure 1. Revised V-model for system developement in the AI era.

to move to a level where purpose, obligation, and stakeholder relevance can still be judged. Reverse engineering is therefore not treated here as a maintenance afterthought, but as a control mechanism. The artifact is translated back into candidate requirements, the human validates semantic equivalence at that abstraction level, and only then are stakeholder-facing media generated. Recent work shows that LLMs can recover formal constraints, user stories, and higher-level descriptions from source code or binaries, but the same literature also reports hallucinations, limited reliability, and strong dependence on prompt design and representation choices [10] [11] [12] [13].

The contribution of the present study is narrower and more operational. It does not ask whether code summarization is possible in general. It asks whether a requirements-mediated path improves the semantic quality of stakeholder-oriented media when compared with direct code-to-media generation, and whether a lightweight knowledge assessment improves audience adaptation. The study is grounded in four proprietary software projects that were not publicly accessible. They were therefore not plausibly part of the pretraining corpora of the media-generation systems under test. That point matters because it reduces the risk that apparent reverse-engineering quality is merely retrieval of memorized project knowledge rather than analysis of unseen artifacts. The resulting comparison is aligned with the revised V-model proposed in Figure 1 and with the research questions that structure the empirical study.

These questions are stated explicitly as follows. **Research Question 1** (RQ1) asks whether AI-supported extraction and human validation of requirements as an intermediate representation lead to higher conceptual accuracy in generated stakeholder-facing media than direct code-to-media generation. **Hypothesis 1** (H1) states that media generated from validated requirements exhibit lower technical hallucination rates and stronger problem-space focus than media generated directly from code. **Research Question 2** (RQ2) asks how effectively an LLM-based implicit knowledge assessment can adapt the linguistic complexity of generated media to non-technical stakeholders. **Hypothesis 2** (H2) states that the three-part steering procedure based on role, goal, and latent knowledge assessment yields outputs whose terminology and complexity are measurably better aligned with the intended user profile than unsteered generation.

A. *Related Work*

The argument developed here intersects four strands of literature. The first concerns LLM-supported software engineering in general. Recent surveys show that LLMs are now applied across a wide range of activities, including code generation, summarization, repair, testing, review, and requirements-related tasks. The field is already broad enough that synthesis work has shifted from cataloguing isolated case studies to organizing application areas, model classes, benchmarks, and evaluation risks [10].

The second strand concerns reverse engineering from executable or source-level artifacts. Siala et al. compare general-purpose and fine-tuned LLMs with model-driven reverse engineering for the extraction of Object Constraint Language (OCL) constraints from Java and Python code. Ouf et al. study the recovery of user stories from C++ code and report strong performance for smaller code units, but also substantial sensitivity to prompt design and code complexity. Pordanesh and Tan examine binary reverse engineering with GPT-4 and identify a similar pattern: promising code understanding, ac-

companied by limits in detailed technical and security analysis [11] [12] [13].

A third strand comes from requirements engineering and traceability. Teleological approaches to requirements analysis have long argued that software specifications must capture not only what a system does, but also why it exists in its organizational setting. Traceability research then formalizes the problem of connecting requirements to later development artifacts and of following those artifacts across the lifecycle. This literature is directly relevant here because a requirements-mediated media pipeline is only defensible if abstraction away from code does not sever the connection to purpose, constraint, and implementation evidence [14] [15] [16].

The fourth strand concerns audience adaptation and source-grounded media generation. Adaptive-learning research consistently treats personalization as a function of prior knowledge, task goals, and observable interaction traces. Knowledge-tracing work adds a more fine-grained model of latent competence, while recent source-grounded systems, such as NotebookLM, show how generated text and audio can be constrained by an explicit source set rather than by open-domain background knowledge alone [17] [18] [19] [20].

What remains largely untested is the combination of these strands in one controlled setup: reverse engineering from complex software artifacts, explicit human validation of the extracted requirements, source-grounded media generation for non-technical stakeholders, and profile-sensitive adaptation of the resulting media. That gap defines the scope of the present experiment.

B. Overview of this paper

Section II describes the experimental workflow, the project corpus, the human validation step for reverse-engineered requirements, and the metrics used to evaluate semantic quality and audience adaptation. Section III reports the comparative results for direct code-to-media generation and requirements-mediated generation, including the central delta analyses and the interpretation of the main score tables. Section IV discusses what can and cannot be concluded from these results with respect to RQ1, RQ2, H1, and H2, and it outlines the next empirical steps.

II. Methodology

The empirical study was implemented as a controlled six-phase pipeline that compares two source-grounding conditions for media generation from software artifacts. The treatment variable is the representation supplied to the media generator: raw code in Group A and human-validated requirements in Group B. Project, role-goal profile, artifact type, language, and evaluation procedure were held constant. Figure 2 summarizes the pipeline within the revised V-model introduced above.

A. Study Design and Experimental Corpus

The experiment covered four heterogeneous software systems. Short project descriptions are given in the following list:

TABLE I. Overview of projects used in experimental setup.

Count	BUBA-Bot	Flight Data Recorder	Legal Chatbot	MSP-Sim
Scanned files	6	269	31	137
Included files	6	129	27	137
Excluded files	0	140	4	0
Shortened files	4	2	1	5
Totel characters	567,843	2,500,000	535,859	2,351,445

- **BUBA-Bot**: a Python-based pipeline for ingestion of the Global Database of Events, Language, and Tone (GDELT), Natural Language Processing (NLP) enrichment, temporal knowledge-graph construction, and Graph-level Joint-Embedding Predictive Architecture (JEPA) based anomaly modeling in geopolitical and monetary-policy analysis.
- **Flight Data Recorder**: embedded firmware in C for the Texas Instruments MSP430 microcontroller family, centered on local monitoring, Global Navigation Satellite System (GNSS) data and sensor integration, and durable Secure Digital (SD) card logging under resource-constrained conditions.
- **Legal Chatbot**: a Python and Azure-based system for drafting German legal opinions from case material, reference archives, section-level retrieval, and template-driven document generation.
- **MSP-Sim**: a simulator for a 16-bit MSP430-like instruction set, implemented in C, with assembler/disassembler logic, execution tracing, memory visualization, and hardware-state inspection.

The corpus was intentionally heterogeneous in language, platform, and code provenance. Flight Data Recorder is embedded C for a microcontroller target. MSP-Sim is also C, but large parts of the codebase were generated from graphical models through Ansys SCADE Display and its KCG code generator; the resulting corpus therefore contains long generated sections and many Open Graphics Library (OpenGL) related calls. The remaining two projects are Python systems. This heterogeneity matters methodologically because it tests whether the reverse-engineering step can recover usable requirement-level abstractions from code that differs strongly in domain, abstraction level, and generation history.

All four projects were proprietary and not publicly accessible. That reduces an obvious contamination threat: a strong result in reverse engineering cannot plausibly be explained by the model merely reproducing memorized public project descriptions. Table I reports the preprocessing statistics for the four repositories. The exact total of 2,500,000 characters for Flight Data Recorder reflects the configured Phase-1 ceiling rather than exhaustion of semantically relevant material. The omitted remainder was dominated by very large source files containing display-font definitions and similar low-semantic boilerplate, so the cap did not materially remove control, logging, or persistence logic from the analysis corpus.

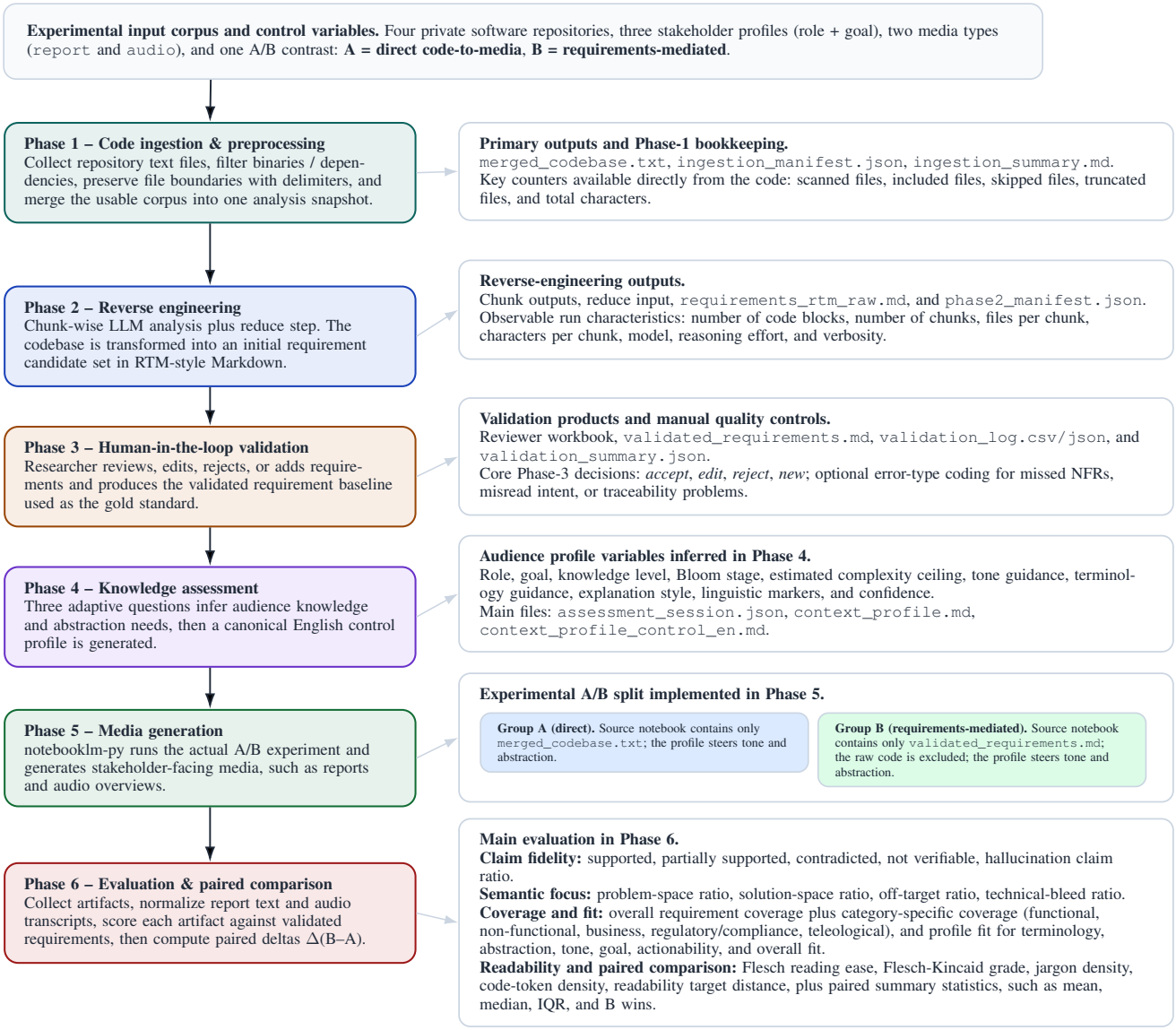

Figure 2. Six-phase pipeline for experimental part of the study.

B. Pipeline Implementation

The full experimental suite is available as open source under the MIT License on Github [21]. The repository was published to make the workflow inspectable and the reported tables reproducible.

Operationally, the pipeline followed six phases.

1) **Phase 1: code ingestion and preprocessing.** The repository walker scans the project tree, skips dependency, cache, build, and binary directories, applies text/binary detection with encoding fallback, and merges the surviving text files into `merged_codebase.txt`. Each file is wrapped in explicit begin/end delimiters and annotated with relative path, byte size, and encoding, so later steps preserve traceability to the original artifact. The implementation also enforces a per-file truncation ceiling of 120,000 characters and a global corpus ceiling of 2,500,000 characters.
2) **Phase 2: chunked reverse engineering.** The merged corpus is parsed back into file blocks and packed into analysis chunks of up to 250,000 characters. Each chunk retains complete file boundaries; an oversized single file becomes its own chunk instead of being split mid-file. Reverse engineering then follows a map-reduce pattern. In the map step, the model extracts candidate requirements for each chunk in five categories—Functional, Non-Functional, Business, Regulatory/Compliance, and Teleological—together with rationale, evidence paths, and confidence markers. In the reduce step, the chunk-level outputs are deduplicated into a draft Requirements Traceability Matrix (RTM) in Markdown format.
3) **Phase 3: human validation.** The draft RTM is ex-

TABLE II. VALIDATION OUTCOMES FOR REQUIREMENTS.

Count	BUBA-Bot	Flight Data Recorder	Legal Chatbot	MSP-Sim
Total	35	20	29	28
Accepted	33	18	23	23
Edited	0	1	5	4
Rejected	2	1	1	1
Missing	0	0	0	0
Included	33	19	28	28

ported to a review workbook with controlled labels for *accept*, *edit*, *reject*, and *new*, plus an inclusion flag for the final RTM. The reviewed workbook is then imported again and materialized as `validated_requirements.md`, which becomes the only requirements source admitted in Group B.

4) **Phase 4: knowledge assessment and profile construction.** A small assessment routine asks for role and goal and then poses three adaptive follow-up questions. From these answers it derives knowledge level, Bloom-stage proxy, complexity ceiling, terminology guidance, tone guidance, explanation style, and linguistic markers. The resulting `context_profile.md` is deliberately code-blind and requirements-blind.
5) **Phase 5: media generation.** The source-grounded generator is Google NotebookLM, used here for audio overviews and structured reports [19] [20] [22] [23]. In the implemented setup, NotebookLM was controlled programmatically through the community-maintained `notebooklm-py` client. To keep the treatment clean, the audience profile was used as steering information rather than as factual source material. Group A uploaded only `merged_codebase.txt`. Group B uploaded only `validated_requirements.md`. Both groups received the same role-goal-specific control profile and the same artifact-type instruction.
6) **Phase 6: normalization, scoring, and comparison.** Reports are evaluated directly. Audio artifacts are first normalized through chunked transcription, because long recordings must be segmented before stable transcription and comparison. The resulting texts are then scored, paired by project, profile, and artifact type, and aggregated into the tables reported in this paper.

This six-phase structure is reflected directly in Figure 2. The figure should be read as a controlled fork after profiling: both branches receive the same communicative target, but they differ in whether the media generator sees raw code or a human-validated requirement abstraction.

C. Human Validation of Reverse-Engineered Requirements

Phase 3 is methodologically central because the reverse-engineering output is treated as a set of candidates, not as ground truth. This follows from the asymmetry discussed in the theoretical section: the mapping from implementation artifacts back to requirements is many-to-many and therefore non-bijective. A human review stage is needed to decide whether the requirement candidates are equivalent, incomplete, overly technical, misclassified, or plainly wrong.

Table II shows the outcome of that review. Across all projects, 112 candidate requirements entered the review workbook, 97 were accepted unchanged, 10 were edited, 5 were rejected, and no missing requirement had to be added after the fact. The downstream corpus used in Group B therefore consists of 108 reviewed requirements. These counts do not yet establish superiority of any media condition. They justify the design choice of treating the requirement layer as a human-checked semantic compression of the artifact rather than as a purely automatic extraction.

D. Knowledge Assessment and Audience Profiles

Phase 4 operationalizes audience adaptation without feeding the media generator extra project facts. The profile is inferred from role, goal, and three adaptive answers. Its function is control, not grounding: it steers terminology, abstraction level, explanation depth, and tone, while all factual statements must still be supported by the uploaded project source.

The experiment instantiated three standardized role-goal profiles spanning business, technical, and learning-oriented reading situations:

- **Marketing Manager / Understand Return on Investment (ROI)**: a business-facing profile focused on value proposition, audience relevance, and commercial implications.
- **Junior Developer / Understand System Architecture**: a technically literate but onboarding-oriented profile focused on structure, major components, and operational logic without full implementation detail.
- **Student / Create Learning Materials**: a learning-oriented profile focused on explanation, conceptual accessibility, and pedagogically useful structuring.

These profiles were chosen because they force the generator to solve three distinct communication problems with the same underlying technical artifact. A good output is therefore not merely factually correct. It must also match the intended audience contract.

E. Evaluation Metrics

Phase 6 combines LLM-judge metrics, lexical coverage metrics, and deterministic readability metrics. The implementation supports heuristic fallbacks, but the evaluation design is centered on source-grounded LLM judging in the now common *LLM-as-a-judge* pattern [24] [25].

The scoring routine first segments each normalized artifact into sentence-level claims; sentences shorter than four tokens are discarded. Each retained sentence is then assigned one support label from {supported, partially supported, contradicted, not verifiable}, one space label from {problem space, solution space, off target}, and a boolean technical-bleed flag. Support labels are judged against the validated requirements, with optional requirement references stored per claim. Claim-level ratios always use the total number of claims as denominator. The *Hallucination Claim Ratio* is defined as the sum of

the *Contradicted Claim Ratio* and the *Not Verifiable Claim Ratio*. This operationalization follows source-grounded work on hallucination that distinguishes contradiction from lack of evidential support, while still treating both as failures of faithful communication [26].

The three space ratios are exhaustive by construction. Every claim receives exactly one of the mutually exclusive labels *problem space*, *solution space*, or *off target*. Their sum is therefore 1.0 for every artifact, aside from rounding in percentage tables. Technical bleed is orthogonal to this triad. A claim may belong to any space class and still contain intrusive implementation detail.

Requirement Coverage is computed separately from the claim annotations and is therefore not an LLM-judge score. For every validated requirement, the script searches the artifact sentences for the strongest lexical match using a weighted token-overlap similarity (0.65 Jaccard overlap and 0.35 containment). A requirement counts as covered when its best sentence-level similarity reaches a threshold of 0.18. The denominator is always the total number of requirements in the relevant set. This rule also applies to the category-specific coverage scores for Functional, Non-Functional, Business, Regulatory/Compliance, and Teleological requirements. The ratio thus measures lexical traceability to the validated RTM, not claim correctness.

Audience-fit scores are produced by a second LLM judge, which compares the artifact text with the Phase-4 control profile and assigns 1–5 scores for terminology, abstraction, tone, goal fit, actionability, and overall fit. By contrast, *Flesch Reading Ease* and *Flesch–Kincaid Grade Level* are deterministic readability measures computed directly from the artifact text [27] [28]. The remaining linguistic indicators are deterministic custom proxies: jargon density is the share of tokens drawn from a technical-term list; code-token density is the frequency of code-like patterns relative to word count; readability target distance measures how far a text's Flesch score falls outside the profile-specific target band implied by the complexity ceiling. In the implementation, these bands are 60–100 for *low*, 30–70 for *medium*, and 0–50 for *high*. All ratios are stored internally as fractions and reported in the paper tables as percentages.

III. Findings

This section first reads the aggregate score distributions in Table III and then turns to the paired contrasts in Tables IV and V and Figure 3. In these tables, ↑ marks a *Higher-Is-Better* metric (HIB), whereas ↓ marks a *Lower-Is-Better* metric (LIB). For the delta tables, positive values are favorable for HIB metrics and negative values are favorable for LIB metrics. The column *B wins* counts, out of the 24 matched A/B comparisons in the experiment, how often the requirements-mediated condition outperforms the direct code-to-media condition under the preferred direction of the respective metric.

A. Aggregate Patterns in the Score Ratios

Table III shows a consistent shift once validated requirements replace raw code as the immediate source for generation. The shift is most visible in the report condition. Mean supported-claim share rises from 4% to 37%, partially supported claims rise from 26% to 40%, and the mean hallucination ratio falls from 70% to 23%. Audio improves as well, but less sharply: supported claims increase from 4% to 15%, partially supported claims from 17% to 22%, and hallucination falls from 79% to 62%.

The same table also clarifies where the broad pattern breaks. Reports outperform audio on most mean fidelity and focus measures, yet not on all of them. Written artifacts contain more technical bleed than audio in both groups, with mean values of 55% versus 28% in Group A and 38% versus 16% in Group B. This is plausible, since reports permit denser technical exposition than podcast-style audio. Solution-space ratio is mixed rather than monotonic: Group B improves audio from 29% to 26%, but increases the mean solution-space share in reports from 44% to 49%. Requirement coverage adds another exception to a simple "reports are better" reading, because audio exceeds report on mean coverage in both groups, from 65% versus 26% in Group A to 72% versus 60% in Group B. The table therefore indicates a broad, but not universal, superiority of Group B and of the report medium.

A second point is easy to miss if one reads only the aggregate hallucination ratio. Contradicted claims are already rare in all four conditions, with mean values between 0% and 2%. The large reduction in hallucination is therefore driven mainly by the drop in *not verifiable* claims, not by a major change in explicit contradiction. In substantive terms, the requirements-mediated condition does not merely suppress a small set of plainly false statements. It yields artifacts whose claims are more often anchored in the validated requirement set.

The extrema temper any overly smooth interpretation. Even in the strongest aggregate condition, Group B reports, the minimum supported-claim share remains 0%, while the maximum hallucination ratio still reaches 100%. The same is true for requirement coverage, whose minimum is 0% in all four group–medium cells. The method improves central tendency. It does not eliminate hard failure cases.

The requirement-coverage floor deserves separate attention. According to the per-project results, the 0% minima in Table III are produced by the Legal Chatbot project. For that project, all role–goal combinations and both media types yield 0%, except the *Marketing Manager / Understand ROI* report condition, which reaches 3.57%. Table II shows that the Legal Chatbot contributed 28 included requirements. The value 3.57% therefore corresponds to exactly one requirement crossing the coverage threshold. This is a cliff-effect result rather than a gradual increase.

Because requirement coverage is computed independently of the claim labels and depends on the best lexical match between each requirement and a single artifact sentence, the

TABLE III. RATIOS (BEST RESULTS PER ROW IN BOLD).

	Group A (codebase only)						Group B (requirements only)					
	Audio			Report			Audio			Report		
Metric (↑: HIB, ↓: LIB)	**Min.**	**Mean**	**Max.**	**Min.**	**Mean**	**Max.**	**Min.**	**Mean**	**Max.**	**Min.**	**Mean**	**Max.**
Supported claim ↑	0%	4%	8%	0%	4%	12%	0%	15%	22%	0%	**37%**	71%
Partially supported claim ↑	3%	17%	27%	0%	26%	52%	4%	22%	42%	0%	**40%**	78%
Contradicted claim ↓	0%	2%	4%	0%	1%	4%	0%	1%	3%	0%	**0%**	2%
Not verifiable claim ↓	60%	77%	94%	38%	69%	100%	38%	61%	95%	3%	**23%**	100%
Hallucination claim ↓	64%	79%	95%	38%	70%	100%	40%	62%	95%	3%	**23%**	100%
Problem space ↑	3%	12%	21%	2%	16%	34%	6%	20%	31%	10%	**27%**	48%
Solution space ↓	9%	29%	51%	8%	44%	75%	7%	**26%**	45%	6%	49%	86%
Off target ↓	40%	59%	88%	10%	40%	85%	35%	54%	87%	3%	**25%**	75%
Technical bleed ↓	7%	28%	45%	10%	55%	91%	2%	**16%**	33%	2%	38%	79%
Requirement coverage ↑	0%	65%	100%	0%	26%	84%	0%	**72%**	100%	0%	60%	100%

TABLE IV. DELTAS (B–A) BY MEDIA TYPE.

Metric (↑: HIB, ↓: LIB)	**Medium: Audio Mean Δ**	**Medium: Report Mean Δ**
Hallucination claim ratio ↓	-16.4%	-46.6%
Problem-space ratio ↑	8.2%	11.0%
Technical-bleed ratio ↓	-12.3%	-17.6%
Requirement coverage ↑	6.4%	33.3%
Profile-fit overall ↑	16.6%	-3.4%
Readability target distance ↓	0.0%	-10.0%

TABLE V. MAIN DELTA (B–A) RESULTS.

Metric (↑: HIB, ↓: LIB)	**Mean Δ**	**B wins**
Hallucination claim ratio ↓	-31.5%	22/24
Problem-space ratio ↑	9.6%	20/24
Technical-bleed ratio ↓	-14.9%	21/24
Requirement coverage ↑	19.9%	15/24
Profile-fit overall ↑	6.6%	17/24
Readability target distance ↓	-5.0%	6/24

anomaly points more plausibly to a measurement bottleneck than to a total semantic collapse of the generated media. Several causes are consistent with the available evidence. The Legal Chatbot corpus mixes English software terminology with specialized German legal concepts. The coverage metric uses no stemming, synonym expansion, bilingual normalization, or semantic embeddings; it rewards literal token overlap. A stakeholder-facing text can therefore be semantically faithful while still scoring 0% if it paraphrases the legal requirement language too aggressively. The sentence-level granularity of the metric intensifies this effect, because a multi-clause legal or compliance requirement may be distributed across several artifact sentences, none of which individually reaches the 0.18 threshold. The project-local nature of the anomaly is revealing here: the other three projects reach at least 12% in every condition, which argues against a general defect of the metric and instead suggests an interaction between this lexical rule and the legal domain.

A further indication comes from the persistence of the floor in both Group A and Group B. Even when validated requirements are the direct generation source, the downstream artifact can still fall to 0% if the medium reformulates those requirements into managerial, educational, or procedural language that preserves intent but not wording. The lone 3.57% case does not overturn this interpretation. It suggests that one sentence retained enough surface vocabulary to cross the threshold. The Legal Chatbot outlier should therefore be read with caution. It signals either threshold sensitivity, domain-specific lexical mismatch, or both. It is weaker evidence for semantic non-coverage than the raw percentage might suggest.

B. Paired Delta Analysis and Hypothesis Assessment

The paired comparisons in Table IV sharpen this picture, they control for medium. For the metrics most directly tied to RQ1 and H1, reports show the larger gains from requirements mediation. Mean hallucination decreases by 46.6 percentage points in reports, compared with 16.4 points in audio. Problem-space focus rises by 11.0 points in reports and by 8.2 points in audio. Technical bleed decreases by 17.6 points in reports and by 12.3 points in audio. Requirement coverage improves by 33.3 points in reports, but only by 6.4 points in audio. The requirements-mediated route helps both media types. Its effect is strongest when the output format permits denser, explicitly structured exposition.

Table V and Figure 3 condense the same result across all 24 matched pairs. The largest overall effect is the reduction in hallucination claim ratio, with a mean delta of -31.5% and 22/24 B wins. Problem-space ratio increases by 9.6 percentage points with 20/24 B wins. Technical bleed decreases by 14.9 points with 21/24 B wins. Requirement coverage increases by 19.9 points with 15/24 B wins. Taken together, these four movements answer RQ1 in the positive direction and provide strong support for H1. The intermediate requirement layer does not merely preserve information. It shifts the generated artifacts toward the validated problem description and away from unverifiable or implementation-heavy content.

The picture is more qualified for RQ2 and H2. The overall profile-fit score improves by 6.6 percentage points, and Group B wins 17 of the 24 matched comparisons. This is not negligible. It indicates that the requirement-mediated path and the Phase-4 control profile often interact productively. Yet the gain is uneven across media. Audio improves markedly, with a mean delta of 16.6 points, whereas reports show a small negative mean delta of -3.4%. Deterministic readability is

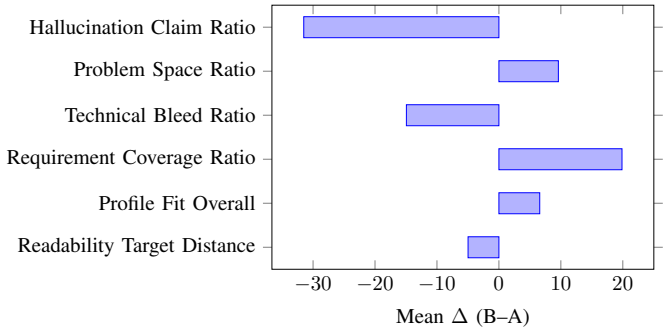

Figure 3. Mean paired deltas (B–A) for the main metrics.

weaker still as evidence for systematic audience adaptation. Readability target distance improves by 10.0 points for reports, remains unchanged on average for audio, and produces only 6/24 B wins overall. The audience-adaptation hypothesis is therefore supported at the level of the judge-based profile-fit score, but only partially at the level of surface readability.

Figure 3 makes the ordering of the effects easy to read. The dominant gains lie in fidelity and semantic focus, not in stylistic tuning. Hallucination reduction is the largest shift, followed by requirement coverage, technical-bleed reduction, and problem-space gain. Profile fit moves in the expected direction, but less strongly. Readability target distance remains the weakest and least stable indicator. This ranking matters for the interpretation of the study as a whole. In the present dataset, the main contribution of reverse-engineered, human-validated requirements is semantic control. Audience alignment improves, but the evidence is more medium-dependent and less robust than the evidence for reduced hallucination and stronger problem-space orientation.

IV. Discussion and Conclusion

This section interprets the findings against the stated research questions and hypotheses, marks the present limits of inference, and sketches the next empirical steps.

A. Discussion

The central result of the study is not a generic quality gain. It is a shift in representational control. Across the matched comparisons, the requirements-mediated condition reduces hallucination claim ratio by 31.5 percentage points, increases problem-space ratio by 9.6 points, reduces technical bleed by 14.9 points, and improves requirement coverage by 19.9 points. These are precisely the outcome dimensions that matter for RQ1. In that sense, the data support a clear answer: introducing reverse-engineered and human-validated requirements as an intermediate representation improves the semantic faithfulness of stakeholder-facing media.

The support for H1 is strong, though not absolute. The pattern is especially clear for reports, where hallucination falls by 46.6 percentage points and requirement coverage rises by 33.3 points. Audio improves as well, but less sharply. This medium asymmetry is plausible. A written report can externalize structure, retain denser conceptual distinctions, and mirror requirement phrasing more directly than a podcast-style narration. The experiment therefore suggests that the proposed V-model adaptation is not medium-neutral in its practical effect. The validated requirement layer helps both artifact types, yet it helps structured written artifacts more.

This matters beyond the narrow benchmark. The four repositories differ substantially in language, architecture, and provenance. One project is embedded C for a microcontroller. One contains large code regions generated from graphical models. Two are Python systems. All are proprietary. The results therefore cannot be reduced to memorized benchmark fragments or to one homogeneous software genre. They indicate that current LLM-based reverse engineering can recover a usable requirement-level abstraction even when the source artifacts are heterogeneous and partly machine-generated. That claim still needs broader confirmation. The present corpus is small.

The evidence for RQ2 and H2 is more qualified. Profile-fit overall improves by 6.6 percentage points, with 17 of 24 comparisons favoring Group B. Readability target distance improves only weakly in the aggregate and does not show a stable win pattern. The by-medium table explains why. Audio gains strongly on profile fit (+16.6 points), while reports decline slightly (-3.4 points). This suggests that the role-goal steering mechanism affected communicative framing and lexical choice, but it did not reliably calibrate textual complexity once the output became more information-dense. In other words, the steering setup appears to help the model decide *what kind of explanation to give*, yet it is less reliable at controlling *how difficult the explanation will be to read*.

That distinction is important for interpretation. Several profile-fit submetrics reported earlier in the pipeline are themselves LLM-judge outputs. They are useful because they capture pragmatic properties that deterministic formulas miss. They also inherit model-based uncertainty. Readability indicators, such as Flesch Reading Ease and Flesch-Kincaid Grade, are deterministic, but they were developed for general prose rather than AI-generated technical communication about software. A modest gain in profile fit combined with an unstable readability result should therefore not be read as a contradiction. It points to a measurement tension: audience alignment is partly semantic and rhetorical, whereas readability formulas primarily register surface complexity.

The Legal Chatbot outlier reinforces that caution. Requirement coverage collapses to 0% in nearly all Legal Chatbot conditions, even though other fidelity indicators do not collapse in the same way. Since requirement coverage is computed lexically rather than through claim judging, the anomaly is best understood as a mismatch between legal paraphrase and surface-overlap scoring, or as an effect of requirement phrasing that is unusually abstract, normatively dense, or terminologically rigid. This does not invalidate the broader pattern, but it constrains the strength of the conclusion. For legally inflected or otherwise specialized domains, lexical coverage should not be treated as a sufficient proxy for semantic coverage without

an additional validation layer.

Taken together, the study supports a narrow but consequential conclusion. The proposed process improves semantic control over generated media and makes a requirement-level validation loop operational. It does not eliminate human judgment. It relocates it. The human role shifts away from inspecting implementation detail and toward validating whether the reconstructed requirement space is equivalent to the intended one. That is the practical contribution of the modified V-model in the age of agentic AI.

B. Next Steps

Several extensions follow directly from the present limitations. The first is scale. Four projects and 24 matched comparisons are sufficient for a structured pilot study, but not for broad generalization. A larger corpus should vary domain, implementation language, code-generation history, and repository size more aggressively. Safety-critical systems, regulated domains, and projects with richer architectural documentation would be especially informative.

The second concerns measurement. The current results already suggest that semantic fidelity and audience adaptation are separable dimensions. Future work should therefore refine the evaluation stack rather than only enlarge it. For requirement coverage, semantic matching methods could be added beside lexical overlap, especially for domains with abstract or formalized language. For audience adaptation, the profile-fit metrics should be complemented by human ratings from readers who actually occupy the target roles, since an LLM judge can approximate stakeholder expectations but cannot fully substitute for them.

The third step is procedural. The present study evaluates one reverse-engineering loop and one downstream generation step. A fuller test of the proposed V-model would examine iterative cycles in which validated requirements are revised, regenerated, and re-checked over several rounds. That would make it possible to study convergence, error propagation, and cost. It would also connect the method more directly to emerging agentic development settings in which implementation, review, and repair alternate rapidly.

A final extension concerns media breadth. Only audio and report artifacts were retained in the final analysis. Other output forms, such as infographics, slide decks, or interactive explainers may behave differently, because they impose different compression rules on the underlying requirement space. The same applies to generator choice. Repeating the experiment with other media-generation systems would test whether the observed gains are process effects or partly tool-specific effects.

C. Summary

This paper examined whether reverse-engineered and human-validated requirements can serve as an effective intermediate layer between software artifacts and AI-generated stakeholder media. The results give a consistent answer for semantic quality. Relative to direct code-to-media generation, the requirements-mediated route reduces hallucination, improves problem-space focus, reduces technical bleed, and raises requirement coverage. These effects are strongest for reports. They remain visible for audio.

The case for audience adaptation is positive but weaker. Role-goal steering improves overall profile fit on average, especially for audio, yet readability control remains unstable. The study therefore supports RQ1 and H1 clearly, while RQ2 and H2 receive only partial support. That asymmetry is informative. In the present setup, the main benefit of the requirement layer is not stylistic polish. It is semantic discipline.

Within those limits, the study contributes both a process proposal and an executable experimental workflow. The open-source pipeline makes the evaluation design inspectable. The proprietary project corpus reduces the risk of memorization-based inflation. The broader implication is straightforward: as agentic systems take over more implementation work, reverse engineering and requirement validation become more central, not less. They become the point at which human oversight remains both feasible and technically meaningful.

ACKNOWLEDGEMENTS

We would like to thank the referees for very useful comments on the original submission. This research did not receive any specific grant from funding agencies in the public, commercial, or not-for-profit sectors.

REFERENCES

[1] Gartner, “Gartner newsroom:gartner identifies the top strategic trends in software engineering for 2025 and beyond.” https://www.gartner.com/en/newsroom/press-releases/2025-07-01-gartner-identifies-the-top-strategic-trends-in-software-engineering-for-2025-and-beyond, 2025. Accessed: 2026-03-28.

[2] J. Bratincevic and R. Taylor and Z. Stone, “Forrester blog:the low-code market could approach $50 billion by 2028.” https://www.forrester.com/blogs/the-low-code-market-could-approach-50-billion-by-2028/, 2024. Accessed: 2026-03-28.

[3] Google Cloud, “Google cloud discover:what is vibe coding?.” https://cloud.google.com/discover/what-is-vibe-coding, 2026. Accessed: 2026-03-28.

[4] OpenAI, “Openai developers:building an ai-native engineering team.” https://developers.openai.com/codex/guides/build-ai-native-engineering-team, 2026. Accessed: 2026-03-28.

[5] Anthropic, “Claude code docs:claude code overview.” https://code.claude.com/docs/en/overview, 2026. Accessed: 2026-03-28.

[6] Ansys, “Product page:ansys scade display.” https://www.ansys.com/products/embedded-software/ansys-scade-display, 2026. Accessed: 2026-03-28.

[7] CodeRabbit, “Coderabbit documentation:ai code review.” https://docs.coderabbit.ai/, 2026. Accessed: 2026-03-28.

[8] GitHub, “Github docs:about github copilot code review.” https://docs.github.com/en/copilot/concepts/agents/code-review/, 2026. Accessed: 2026-03-28.

[9] Greptile, “Greptile docs:overview – what is greptile?.” https://www.greptile.com/docs/introduction, 2026. Accessed: 2026-03-28.

[10] Q. Zhang, C. Fang, Y. Xie, Y. Zhang, Y. Yang, *et al.*, “A survey on large language models for software engineering,” *ArXiv*, vol. abs/2312.15223, 2023.

[11] H. Siala and K. Lano, “A comparison of large language models and model-driven reverse engineering for reverse engineering,” *Frontiers in Computer Science*, vol. Volume 7 - 2025, 2025.

[12] M. Ouf, H. Li, M. Zhang, and M. Guizani, "Reverse engineering user stories from code using large language models," *2025 IEEE International Conference on Collaborative Advances in Software and COmputiNg (CASCON)*, pp. 504–509, 2025.
[13] S. Pordanesh and B. Tan, "Exploring the efficacy of large language models (gpt-4) in binary reverse engineering," *ArXiv*, vol. abs/2406.06637, 2024.
[14] P. Loucopoulos and E. Kavakli, "Enterprise modelling and the teleological approach to requirements engineering," *International Journal of Intelligent and Cooperative Information Systems*, vol. 4, no. 1, pp. 45–79, 1995.
[15] B. Ramesh, C. Stubbs, T. Powers, and M. Edwards, "Requirements traceability: Theory and practice," *Annals of Software Engineering*, vol. 3, pp. 397–415, 1997.
[16] S. Winkler and J. Pilgrim, "A survey of traceability in requirements engineering and model development," *Software and System Modeling*, vol. 9, pp. 529–565, 09 2010.
[17] I. Gligorea, M. Cioca, R. Oancea, A. T. Gorski, H. Gorski, *et al.*, "Adaptive learning using artificial intelligence in e-learning: A literature review," *Education Sciences*, 2023.
[18] Y. Lu, L. Tong, and A. Cheng, "Advanced knowledge tracing: Incorporating process data and curricula information via an attention-based framework for accuracy and interpretability," *Journal of Educational Data Mining*, vol. 16, p. 58–84, Oct. 2024.
[19] R. and S. Johnson, "Google blog:notebooklm: How to try google's experimental ai-first notebook." https://blog.google/innovation-and-ai/technology/ai/notebooklm-google-ai/, 2023. Accessed: 2026-03-28.
[20] B. Wang, "Google blog:notebooklm now lets you listen to a conversation about your sources." https://blog.google/innovation-and-ai/products/notebooklm-audio-overviews/, 2024. Accessed: 2026-03-28.
[21] M. Harter, "Aimedia experiment suite." https://github.com/matthias-harter/aimedia_experiment_suite, 2026. Accessed: 2026-03-27.
[22] M. Paul and D. Dhami, "Google blog:6 ways to use notebooklm to master any subject." https://blog.google/innovation-and-ai/models-and-research/google-labs/notebooklm-student-features/, 2025. Accessed: 2026-03-28.
[23] T. Lin, "notebooklm-py." https://github.com/teng-lin/notebooklm-py, 2026. Accessed: 2026-03-27.
[24] Y. Liu, D. Iter, Y. Xu, S. Wang, R. Xu, *et al.*, "G-eval: Nlg evaluation using gpt-4 with better human alignment," *ArXiv*, vol. abs/2303.16634, 2023.
[25] H. Li, Q. Dong, J. Chen, H. Su, Y. Zhou, Q. Ai, *et al.*, "Llms-as-judges: A comprehensive survey on llm-based evaluation methods," *ArXiv*, vol. abs/2412.05579, 2024.
[26] Z. Ji, N. Lee, R. Frieske, T. Yu, D. Su, *et al.*, "Survey of hallucination in natural language generation," *ACM Computing Surveys*, vol. 55, p. 1–38, Mar. 2023.
[27] R. F. Flesch, "A new readability yardstick.," *The Journal of applied psychology*, vol. 32 3, pp. 221–33, 1948.
[28] J. P. Kincaid, J. Fishburne, Robert P., R. L. Rogers, and B. S. Chissom, "Derivation of new readability formulas (automated readability index, fog count and flesch reading ease formula) for navy enlisted personnel," Research Branch Report 8-75, Institute for Simulation and Training, University of Central Florida, Millington, TN, 1975.