
# Does ISO-Grounded NFR Specification Improve LLM Code Generation? A Comparison of Rich and Structured Interventions against a Natural-Language Baseline

João Pedro Monteiro Pereira
Centro de Informática, Universidade Federal de Pernambuco
Recife, PE, Brazil
jpmp2@cin.ufpe.br

Vinicius Cardoso Garcia
Centro de Informática, Universidade Federal de Pernambuco
Recife, PE, Brazil
vcg@cin.ufpe.br

## Abstract

In LLM-based code generation, Non-Functional Requirements (NFRs) are often specified as terse one-line phrases. We ask whether grounding those specifications in ISO/IEC 25010:2023, either as rich natural-language prose (NL-rich) or as structured JSON (Structured), improves code generated on HumanEval/HumanEval-ET compared to a RobuNFR-style one-line baseline (NL-simple). We evaluate four NFRs (performance, error handling, code smell, readability) with ten prompt variations per condition under a fixed model snapshot and paired non-parametric analysis. **Primary finding:** ISO-grounded enrichment improves static quality proxies (unreadability density falls across all four NFRs (e.g., Performance 0.88→0.69 for NL-rich)) and reduces sensitivity to prompt wording, but does not reliably improve functional correctness; for error handling, extended-test pass rate decreases, suggesting tension between defensive coding patterns and exact-output benchmarks. **Secondary finding:** when ISO content is held constant, NL-rich and Structured differ negligibly in correctness ($|\delta| \leq 0.023$), indicating that semantic content matters more than JSON-vs-prose format. Practitioners should invest in standard-grounded NFR content rather than serialization form. A fully traceable replication package is provided.

## Keywords

Non-Functional Requirements, Large Language Models, Code Generation, Prompt Engineering, Software Quality, ISO/IEC 25010, Empirical Software Engineering

## 1 Introduction

Large Language Models (LLMs) are now routinely used to generate source code from natural-language descriptions [2, 3]. While most benchmarks focus on *functional* correctness, real software must also satisfy *Non-Functional Requirements* (NFRs) such as performance efficiency, reliability, and maintainability [6, 14]. Functional correctness is commonly summarized as **Pass@1**: the fraction of benchmark tasks solved correctly on the first generation attempt. Recent work evaluates how well LLMs honor NFRs and how *robust* outputs are when the same requirement is re-worded. RobuNFR [10] shows that adding an NFR to a prompt tends to reduce Pass@1 and increases standard deviation across surface variations of the same requirement.

**Problem.** In practice, NFRs are often expressed as short, ambiguous natural-language phrases. RobuNFR adopts this style (one line per NFR). Teams that care about software quality may instead document NFRs using product-quality models such as ISO/IEC 25010:2023, either as rich prose or as structured artifacts consumed by architecture and MDE tooling. It remains unclear whether such *enrichment* helps LLMs produce better code, or whether only the *format* (prose vs. JSON) matters.

**Gap.** Prior work establishes that NFRs affect generation and that prompt variation matters [10], and that ISO-grounded quality characteristics are relevant to LLM-generated code [14]. What is missing is a controlled comparison of *enrichment level* (NL-simple vs. ISO-grounded content) and, secondarily, of *representation form* (NL-rich vs. Structured) under fixed ISO content.

**Investigation.** We evaluate two ISO/IEC 25010:2023-grounded interventions against a RobuNFR-style **NL-simple baseline** (one-line NFR per task): (i) **NL-rich**, a detailed natural-language paragraph; and (ii) **Structured**, the same ISO content serialized as JSON. All three conditions use `gpt-5.4-2026-03-05`; the NL-simple baseline was collected in an earlier batch (Apr–May 2026) and was not re-run when interventions executed (June 2026). Ten prompt variations per condition, identical functional task clauses, and HumanEval/HumanEval-ET yield paired per-problem statistics for four NFRs: performance, error handling, code smell, and readability.

We investigate four research questions, ordered by priority:

- **RQ1 (Correctness vs. baseline).** Do NL-rich or Structured improve functional correctness (Pass@1, ET-Pass@1) over NL-simple?
- **RQ2 (NFR quality vs. baseline).** Do they improve NFR-specific code quality (exception, code-smell, and unreadability densities per ten LOC)?
- **RQ3 (Robustness vs. baseline).** Do they reduce sensitivity across ten prompt variations (STDEV) relative to NL-simple?
- **RQ4 (Form; secondary).** When ISO content is held constant, does representation form (NL-rich vs. Structured) affect correctness, quality, or robustness?

**Preview of answer.** Enrichment improves *quality proxies* and *robustness* relative to NL-simple, but not functional correctness (and may harm Error Handling ET-Pass@1). When ISO content is fixed, prose and JSON behave similarly. The scientific story is therefore: *what* you specify (ISO-grounded enrichment) matters more than *how* you serialize it (NL-rich vs. JSON).

Our contributions are: (i) paired statistical comparisons of both interventions against NL-simple (primary); (ii) a content-controlled form comparison (secondary); (iii) evidence that enrichment narrows prompt sensitivity despite shorter, less lexically diverse prompts;

and (iv) a transparent replication package with traceability from every reported number to its source file.

The remainder of the paper is organized as follows. Section 2 summarizes background; Section 3 positions related work; Section 4 describes the method; Section 5 presents results; Section 6 discusses implications; Section 7 lists threats to validity; Section 8 concludes.

## 2 Background

### 2.1 Non-Functional Requirements and ISO/IEC 25010

NFRs constrain *how* a system behaves rather than *what* it computes. ISO/IEC 25010:2023 [8] defines a product-quality model with characteristics including *Performance Efficiency* (e.g., time behavior, resource utilization), *Reliability* (e.g., fault tolerance), and *Maintainability* (e.g., analysability, modifiability). ISO/IEC 25002:2024 [9] provides the conceptual overview and guidance on how such quality models are used. We use these standards as an external, citable source of NFR *content*, so that NL-rich and Structured encode the same standard-grounded information and differ only in representation form. This design lets us separate *enrichment* (adding ISO-grounded constraints and acceptance criteria beyond a one-line phrase) from *format* (prose vs. JSON).

### 2.2 LLM Code Generation and Its Evaluation

Functional correctness of generated code is commonly measured with Pass@1 on HumanEval [3] and MBPP [2]. Because the original test suites can be weak, EvalPlus [11] augments them with additional inputs (HumanEval+). We additionally report ET-Pass@1 on HumanEval-ET extended oracles, following the RobuNFR evaluation protocol [10], to reduce false positives. We report both Pass@1 and ET-Pass@1. Functional metrics alone cannot establish NFR satisfaction; we therefore complement them with NFR-aligned static-analysis densities (next subsection) and prompt-variation robustness (Section 5).

### 2.3 Code-Quality Metrics as Densities

Following RobuNFR [10], NFR-related quality is measured with static analysis normalized by code size: *code-smell density* from Pylint [12] *Refactor* messages and *unreadability density* from Pylint *Convention* messages, both per ten lines of code (LOC); *exception density* as exception-handling statements per ten LOC; and execution time as the mean over repeated runs. Normalizing per LOC is essential because richer or structured prompts can produce longer code, and raw issue counts would otherwise confound size with quality. LOC is computed by the evaluation pipeline.

We selected these proxies because they (i) are automatable at scale across 164×10 completions per condition, (ii) map reasonably to Maintainability and Reliability wording in our ISO objects, and (iii) remain comparable to RobuNFR's reporting style. They do not measure all ISO/IEC 25010 characteristics (e.g., security, compatibility); our four NFRs focus on performance efficiency, fault tolerance, and maintainability-related concerns that admit static or lightweight dynamic signals on HumanEval-sized programs.

## 3 Related Work

**Robustness of NFR-aware generation.** RobuNFR [10] evaluates LLM robustness across four NFR dimensions (design, readability, reliability, performance) using prompt variation, regression testing, and diverse workflows, and reports that including NFRs reduces Pass@1 (by up to 39%) while increasing standard deviation across re-wordings. RobuNFR's NFR prompts are intentionally terse (one line), mirroring how practitioners sometimes specify quality in ad hoc natural language. Our study extends this methodology with ISO-grounded *enrichment* as the primary factor and treats representation *form* as secondary.

**Requirement-aware generation.** ArchCode [6] organizes functional and non-functional requirements from textual descriptions via in-context learning and introduces HumanEval-NFR, the first benchmark to evaluate NFRs alongside functional requirements. ArchCode focuses on *improving* requirement satisfaction through retrieval and organization. We instead ask a complementary question: given fixed functional tasks, does *how much* and *how* NFR content is specified change outcomes when the model and benchmark are held constant?

**Benchmark and protocol choice (RobuNFR vs. HumanEval-NFR).** We use HumanEval/HumanEval-ET with RobuNFR's ten-variation protocol rather than HumanEval-NFR for three reasons: (i) our primary factor is ISO-grounded *enrichment* against a one-line baseline and a secondary *form* contrast with content held constant (not requirement classification or retrieval from unstructured text, which HumanEval-NFR targets); (ii) RobuNFR's terse NFR prompts align with our NL-simple baseline and document sensitivity under the same functional scaffold; (iii) HumanEval-ET supplies extended functional oracles across conditions without changing the task distribution. We cite RobuNFR as a *methodological reference* and collect our own NL-simple baseline under `gpt-5.4-2026-03-05`; replication on HumanEval-NFR is future work.

**Quality characteristics of generated code.** A recent study on quality assurance of LLM-generated code organizes the analysis around ISO/IEC 25010 quality characteristics and combines a literature review, practitioner workshops, and an empirical study [14]. This corroborates the relevance of standard-grounded NFRs but does not compare enrichment level against a one-line baseline, nor does it hold ISO content constant while varying form.

**Structured vs. natural-language requirements.** In component-based and model-driven workflows, NFRs often appear as structured attributes (e.g., JSON, DSLs, architecture decision records) linked to quality models. A natural hypothesis is that LLMs might prefer structured input. Our RQ4 directly tests this under content control; the empirical answer is that format alone does not move correctness once ISO content is fixed.

**NFR specification quality and ambiguity.** Requirements-engineering research consistently finds that NFRs are hard to state precisely and are frequently left ambiguous. A controlled experiment shows that NL requirements-quality defects (notably ambiguous pronouns) propagate into downstream activities such as domain modeling [5]; a survey shows that NFRs in particular tend to be specified late, in uncertain and unstable terms, and revised throughout a project [15]. Almonte et al. [1] use LLMs to derive NFRs grounded in ISO/IEC 25010:2023 from functional requirements and report strong

agreement with expert assessments. Together these works establish (i) that underspecified requirement phrasing, including for NFRs, is a real and costly problem and (ii) that ISO/IEC 25010 is a natural backbone for making NFRs explicit. We differ in goal and design: rather than *generating* or *classifying* NFRs, we treat ISO-grounded content as a controlled *prompt-input manipulation* and measure its effect on the generated code (correctness, static-quality densities, and robustness) against a terse NL-simple baseline, directly connecting the well-documented difficulty of operationalizing NFRs in measurable terms to a concrete generation outcome (Section 6.6).

**Research gap.** Prior work shows that (i) NFRs affect LLM outputs, (ii) prompt variation matters, and (iii) ISO quality models are relevant. Missing is a paired, per-problem comparison of *enrichment* (NL-simple vs. ISO-grounded interventions) with a secondary, content-controlled form contrast. Our primary contribution fills the first gap; RQ4 fills the second.

**Evaluation infrastructure.** HumanEval [3], MBPP [2], and EvalPlus [11] provide functional-correctness scaffolding; HumanEval-ET extended oracles follow RobuNFR [10]. Pylint [12] supplies the static-analysis metrics we reuse.

# 4 Research Method

## 4.1 Overview and Experimental Conditions

We compare two **interventions** against a **NL-simple baseline**:

- **NL-simple (baseline)**: the RobuNFR-style one-line NFR phrase per task (collected in this study, same model snapshot).
- **NL-rich (intervention)**: ISO/IEC 25010 content written as a rich natural-language paragraph.
- **Structured (intervention)**: the *same* ISO content serialized as JSON (attribute, intent, ISO mapping, constraints, acceptance criteria).

NL-rich and Structured share identical ISO-grounded content; only representation form differs between them. All three conditions use the same model (`gpt-5.4-2026-03-05`), decoding configuration (temperature 0), HumanEval problem set, and evaluation pipeline. We additionally report a **Function-Only** (no-NFR) baseline (the bare HumanEval stub with no quality clause) as an NFR-independent context row in Table 1; it lets the reader judge whether the one-line NL-simple baseline already shifts quality relative to specifying no NFR at all. Function-Only is a context reference, not part of the paired RQ1–RQ4 comparisons.

Figure 1 summarizes the study design at a high level. The **primary axis** contrasts each intervention against NL-simple (RQ1–RQ3). The **secondary axis** compares NL-rich vs. Structured with ISO content held constant (RQ4). Both axes reuse RobuNFR's ten-variation protocol and the same functional task clause across conditions.

## 4.2 Illustrative Prompt Contrast (Code Smell NFR)

To make the manipulation concrete, Listing 2 sketches the three NFR blocks for one NFR (surface wording varies across the ten prompt templates; ISO fields are identical between NL-rich and Structured). The functional prefix (*"complete the following code:"* plus the HumanEval stub) is identical in all conditions.

The NL-simple phrase is short and leaves interpretation to the model. NL-rich and Structured spell out ISO characteristics, explicit constraints, and acceptance criteria tied to the quality attribute (never to passing HumanEval tests).

**Study design (conceptual).**
Functional task (fixed) + NFR block (manipulated).
**Primary:** NL-rich vs. NL-simple; Structured vs. NL-simple.
**Secondary:** Structured vs. NL-rich (same ISO object).
Outcomes: Pass@1, ET-Pass@1, density metrics, STDEV across 10 variations.
Model: `gpt-5.4-2026-03-05`, $T$=0; benchmark: HumanEval/ET ($n$=164).

**Figure 1: Experimental factors and comparison hierarchy. Primary comparisons evaluate enrichment; secondary comparison evaluates form under fixed content.**

```
NL-simple:
  Avoid code smells in the implementation.

NL-rich (excerpt):
  Maintainability (ISO/IEC 25010): modularity and
  analysability. Avoid refactor-worthy patterns; keep
  functions cohesive; limit nesting; ... [constraints
  and acceptance criteria for smell-related quality]

Structured (excerpt):
  {"attribute":"code_smell","iso_25010":{...},
   "constraints":[...],"acceptance_criteria":[...]}
```

**Figure 2: Illustrative NFR blocks for the Code Smell condition (abbreviated). Full templates are in the replication package.**

## 4.3 Holding Content Constant; Varying Only Form

To make the rich-NL and structured conditions content-equivalent, both are produced from a single source object per NFR (attribute, intent, ISO characteristic/sub-characteristics, constraints, acceptance criteria). The *functional task clause* (*"complete the following code:"*) is byte-for-byte identical across all conditions, so the only manipulated difference is whether the NFR block is prose or JSON. The acceptance criteria are restricted to the quality attribute and never instruct the model to "pass the tests", preventing a leakage confound on Pass@1. For each condition we create ten prompt variations (varying the surface realization) to measure prompt sensitivity, mirroring RobuNFR's ten-variation protocol.

## 4.4 ISO Mapping

Each NFR is mapped to ISO/IEC 25010:2023 as follows: *performance* → Performance Efficiency (time behavior, resource utilization); *error handling* → Reliability (fault tolerance) with relevant exception-handling practice; *code smell* and *readability* → Maintainability

(modularity/analysability and analysability/modifiability, respectively). The mapping is an explicit design decision documented in the package; we treat its subjectivity as a construct threat (Section 7).

For each NFR, the shared ISO object includes: (i) a short *intent* statement aligned with the sub-characteristic; (ii) two to four *constraints* phrased as imperative requirements on generated code; and (iii) *acceptance criteria* that remain attribute-specific (e.g., limiting nesting depth for readability) and never reference HumanEval test passage. NL-rich renders these fields as connected prose; Structured emits the same fields as JSON keys. Ten prompt *variations* paraphrase surface wording (synonyms, clause order) while preserving the ISO object, following RobuNFR's sensitivity protocol.

## 4.5 Model, Benchmark, and Generation

We use the model `gpt-5.4-2026-03-05`with temperature 0 (greedy), the only decoding configuration in the study. The NL-simple baseline was collected in an earlier batch within this study (Apr–May 2026); the NL-rich and Structured interventions followed in June 2026. All conditions use the same pinned snapshot and temperature 0. Because batch timing differs, we report separate *post hoc* snapshot-stability checks (Section 4.6) rather than treating the batches as interchangeable without qualification. OpenAI's snapshot policy states that model *snapshots* lock a specific version for consistent behavior [13]; we cite it only as secondary support. Generation targets the HumanEval problem set (164 tasks) via EvalPlus [11]. For each (NFR, condition) we generate code for all ten prompt variations, yielding the per-problem completions analyzed below. Generation concurrency is an infrastructure parameter only and does not affect prompts, model, temperature, or any metric.

## 4.6 Snapshot Stability Check (Batch Mitigation)

Because the NL-simple baseline and the June interventions were collected in separate batches, we ran *post hoc* stability checks on NL-simple prompt0 with the same `gpt-5.4-2026-03-05` snapshot and pipeline, comparing per-problem Pass@1 against the April–May baseline using paired Wilcoxon tests (10,000 bootstrap resamples for mean-difference 95% CIs; scripts in `results/w1-stability`).

**Full-set check (Performance, $n = 164$).** We re-executed NL-simple prompt0 on all 164 HumanEval tasks in August 2026. Mean Pass@1 was 0.93 (Apr–May) vs. 0.92 (August): 97.6% per-task agreement (one task improved, three worsened), paired Wilcoxon $p = 0.424$, Cliff's $\delta = 0.01$ (negligible), and bootstrap 95% CI for the mean Pass@1 difference $[-0.04, +0.01]$, which *includes zero*. ET-Pass@1 was likewise non-significant (0.84 vs. 0.84; $p = 0.773$). We find no strong evidence of snapshot drift on the complete Performance subset.

**Pilot subset ($n = 30$, June 2026).** Earlier, we re-ran NL-simple prompt0 on a fixed stratified subset (seed 42; pass/fail balanced from the Performance baseline) for Performance and Error Handling in June 2026. On this subset, Performance Pass@1 was 0.63 vs. 0.77 ($p = 0.072$; bootstrap CI $[+0.03, +0.27]$, excluding zero), while Error Handling was stable (0.83 vs. 0.87; $p = 1.0$). Because the subset over-samples failing tasks (11 failures exist in the full baseline), its absolute Pass@1 rates are much lower than on all 164 tasks and should not be read as contradicting the full-set August check.

**Interpretation.** Non-concurrent collection between the April–May baseline and the June interventions remains an internal-validity concern, but the August full-set re-run suggests that snapshot drift on Performance NL-simple prompt0 is unlikely to explain the primary comparisons. We still report primary statistics with their original batch timestamps and interpret the pilot subset as underpowered, exploratory evidence. Concurrent re-collection of the full NL-simple baseline across all NFRs and prompt variations is listed as future work.

## 4.7 Metrics

We report: **Pass@1** and **ET-Pass@1** (functional correctness on HumanEval and HumanEval-ET); **exception density**, **code-smell density**, and **unreadability density** (issues per ten LOC, from Pylint Refactor/Convention and exception statements); **LOC**; and **execution time** (mean over five runs per problem, and the ET variant). To avoid survivorship bias, NFR-quality metrics are compared over a shared set of problems and paired per problem (Section 4.8).

## 4.8 Statistical Analysis

The **primary contrasts** are each intervention (**NL-rich**, **Structured**) versus **NL-simple**, using per-problem paired Wilcoxon signed-rank tests [16], Cliff's delta [4] ($\delta > 0$ favors the intervention), and Holm–Bonferroni correction [7] within each (NFR, comparison, metric family). We choose non-parametric paired tests because Pass@1 and density metrics are bounded and often skewed across problems; Wilcoxon respects pairing (same HumanEval task under two prompt conditions) without assuming normality of differences. Cliff's $\delta$ complements $p$-values: several correctness contrasts are statistically significant yet negligible in magnitude (e.g., Error Handling ET-Pass@1), so we report both. Holm–Bonferroni controls family-wise error within each (NFR, comparison, metric family) rather than globally across all tables, matching our prespecified contrast structure.

The **secondary contrast** (RQ4) is NL-rich vs. Structured with content held constant. Baseline per-problem vectors are loaded from NL-simple evaluation JSONs collected for `gpt-5.4-2026-03-05`; intervention vectors from the June 2026 runs. Per-problem density vectors (unreadability, code-smell, exception) are derived from each JSON's Pylint buckets and exception statements normalized by LOC (Section 4.7). **Robustness** (RQ3): Table 5 reports condition-level STDEV of aggregate metrics across ten prompt variations (descriptive). We additionally apply paired Wilcoxon tests to *per-problem* STDEV (for each task, the sample STDEV of a metric across its ten prompt variations), comparing each intervention to NL-simple within each (NFR, comparison) family of four metrics, with Holm correction (Section 8). Mean pairwise Jaccard distance remains descriptive; lower diversity on enriched prompts partly reflects shared ISO template boilerplate, not semantic equivalence alone.

## 4.9 Reproducibility

All configurations, prompts, generated code, evaluation outputs, and analysis scripts are released in a replication package (Section 8). Any execution-time change required to run the pipeline is documented in the package's `RUNTIME_NOTES.md`. A traceability map

(`results_numbers.json`) links each table cell and $p$-value to the originating evaluation JSON or Excel summary, supporting independent verification without re-running generation.

## 5 Results

All numbers below come from the generated result files and are reproduced, with their sources, in the replication package. We generated and evaluated all three conditions for `gpt-5.4-2026-03-05`: NL-simple in an earlier batch (Apr–May 2026) and NL-rich and Structured in June 2026. Every (NFR, condition) cell comprises ten prompt variations over the 164 HumanEval problems. Table 1 reports per-condition means; Table 2 the **primary** paired tests for correctness and time (each intervention vs. NL-simple); Table 3 the **RQ2** paired density tests vs. NL-simple; Table 4 the **secondary** form-only contrast; and Table 5 robustness and prompt diversity.

### 5.1 RQ1: Functional Correctness vs. NL-Simple

Neither intervention reliably *improves* functional correctness over the NL-simple baseline (Table 2). For Performance, Code Smell, and Readability, all Pass@1 and ET-Pass@1 comparisons have negligible Cliff's $\delta$ ($|\delta| \leq 0.033$) and are non-significant after Holm correction. At the aggregate level (Table 1), mean Pass@1 differs by at most 1.4 percentage points from NL-simple for any intervention (e.g., Performance NL-rich 94.5 vs. NL-simple 93.1).

The exception is **Error Handling**, where enrichment *hurts* extended-test correctness: NL-rich ET-Pass@1 drops from 81.0 to 76.0 with $p_{Holm} = 0.001$ ($\delta = -0.027$, negligible magnitude but significant); Structured ET-Pass@1 drops to 78.4 with $p_{Holm} = 0.048$. NL-rich Pass@1 also falls (95.1 → 92.3; raw $p = 0.025$, negligible $\delta$). At roughly five percentage points absolute, the ET-Pass@1 decrease is modest in pass rate but systematic across problems after pairing. We interpret this as the richer fault-tolerance specification eliciting exception patterns that conflict with HumanEval's functional tests on some problems, consistent with RobuNFR's observation that NFRs can reduce Pass@1 [10]. HumanEval's oracles emphasize exact input-output behavior on small functions; they rarely reward defensive try/except structure unless the reference solution includes it.

### 5.2 RQ2: NFR-Specific Code Quality vs. NL-Simple

Where interventions show clear movement is in **static-analysis quality densities** (Table 1; paired tests in Table 3). **Unreadability density** falls markedly versus NL-simple for every NFR in both aggregate means and paired per-problem tests (all eight intervention comparisons have $p_{Holm} < 0.05$ with negative $\delta$): Performance 0.88 → 0.69 (NL-rich, −22% relative) and 0.79 (Structured, −10%); Error Handling 0.57 → 0.42/0.43 (−26%/−25%); Code Smell 0.67 → 0.53/0.56; Readability 0.60 → 0.55/0.56. Because densities are normalized per ten LOC, these shifts are not artifacts of shorter or longer outputs alone: LOC does increase for some interventions (e.g., Error Handling NL-rich 4244 → 4692 mean LOC), which would tend to inflate raw issue counts if unnormalized.

For context, the **Function-Only** (no-NFR) row in Table 1 is **descriptive context only** (not part of paired RQ1–RQ4): it shows higher unreadability density (1.85 per ten LOC) than any NFR condition despite competitive Pass@1 (94.5%). We do not test this contrast inferentially.

For the Code Smell NFR, **code-smell density** drops from 0.019 (NL-simple) to 0.011 (NL-rich) at aggregate level (42% relative reduction), with Structured at 0.012; the paired per-problem test for NL-rich vs. NL-simple is borderline (raw $p = 0.049$, $p_{Holm} = 0.10$, negligible $\delta$). Exception density rises for Error Handling (0.99 → 1.18), significant in paired tests ($p_{Holm} < 0.05$, positive $\delta$). We treat this increase as **ambiguous evidence** rather than a clear quality gain: a higher exception density is consistent with the NFR's fault-tolerance intent (more explicit input validation and error paths), but it is also the most likely mechanism behind the ET-Pass@1 drop reported in Section 5.1, because added `try`/`except` and `raise` statements can conflict with HumanEval's exact-output oracles. The same signal can therefore indicate either better defensive design or reduced benchmark compatibility, and the static metric alone cannot disambiguate the two. For Error Handling, paired tests also show significant code-smell reductions ($p_{Holm} < 0.05$).

For **execution time**, both interventions differ strongly from NL-simple in paired tests (large $\delta$ in all NFRs), with direction depending on the NFR (e.g., Error Handling interventions are faster ($\delta \approx -1.0$) while Performance interventions are slower ($\delta = 1.0$). Because execution time depends on hardware and is noisy, we report these only as internal deltas (Section 7).

### 5.3 RQ3: Robustness vs. NL-Simple

Table 5 reports condition-level STDEV across ten prompt variations (descriptive). Pass@1 STDEV falls for Performance (NL-simple 0.0158 to NL-rich 0.0057; 64% lower) and Error Handling (0.0151 to 0.0092/0.0087), but paired Wilcoxon tests on *per-problem* Pass@1 STDEV do not reach significance after Holm correction (e.g., Performance NL-rich: $p = 0.044$, $p_{\text{Holm}} = 0.13$).

Unreadability STDEV shows a clearer pattern: aggregate reductions (e.g., Performance 0.123 to ≈ 0.037) are confirmed inferentially: all eight intervention–baseline contrasts (four NFRs × NL-rich and Structured) yield $p_{\text{Holm}} < 0.05$ with negative Cliff's $\delta$ on per-problem unreadability STDEV. Several code-smell and exception STDEV contrasts are likewise significant (full inferential results in the replication package; Section 8). Code Smell Pass@1 STDEV is mixed at the aggregate level (NL-rich slightly higher than NL-simple); Readability Pass@1 STDEV stays low across conditions (97.1% baseline Pass@1).

Intervention prompts are *shorter and less textually diverse* than NL-simple (Jaccard 0.10–0.13 vs. 0.44–0.59), yet exhibit lower unreadability STDEV under both descriptive and inferential summaries; the Jaccard gap partly reflects fixed ISO template text. With RQ2, enrichment appears to steer models toward implementations with similar test outcomes but lower static-quality sensitivity to re-wording.

### 5.4 RQ4: Form (Secondary; NL-Rich vs. Structured)

When ISO content is held constant, representation form has *no statistically significant effect on functional correctness* (Table 4): every Pass@1 and ET-Pass@1 comparison has negligible $\delta$ ($|\delta| \leq 0.023$) and is non-significant after Holm correction (smallest $p_{Holm} = 0.080$

**Table 1: Mean over ten prompt variations per (NFR, condition) on HumanEval. Densities are issues per ten LOC. The Function-Only (no-NFR) row is an NFR-independent context baseline (bare stub, no quality clause). NL-s=NL-simple baseline, NL-r=NL-rich, St=Structured. Source: aggregate Excel files (see `results_numbers.json`).**

| NFR | Cond. | Pass@1 (%) | ET-Pass@1 (%) | Exc. density | Smell density | Unread. density | LOC | Time (s) |
|---|---|---|---|---|---|---|---|---|
| Function-Only (no NFR) | | 94.5 | 83.4 | 0.02 | 0.06 | 1.85 | 1694.7 | 0.06 |
| Performance | NL-s | 93.1 | 82.8 | 0.03 | 0.04 | 0.88 | 2736.3 | 0.04 |
| | NL-r | 94.5 | 82.4 | 0.05 | 0.04 | 0.69 | 3087.4 | 0.08 |
| | St | 93.4 | 81.8 | 0.05 | 0.04 | 0.79 | 2823.0 | 0.08 |
| Error Handling | NL-s | 95.1 | 81.0 | 0.99 | 0.05 | 0.57 | 4244.4 | 0.17 |
| | NL-r | 92.3 | 76.0 | 1.18 | 0.03 | 0.42 | 4692.0 | 0.06 |
| | St | 93.6 | 78.4 | 1.16 | 0.03 | 0.43 | 4540.8 | 0.06 |
| Code Smell | NL-s | 96.8 | 84.6 | 0.11 | 0.02 | 0.67 | 2850.5 | 0.05 |
| | NL-r | 96.3 | 84.2 | 0.13 | 0.01 | 0.53 | 3679.9 | 0.06 |
| | St | 96.2 | 84.0 | 0.12 | 0.01 | 0.56 | 3360.0 | 0.06 |
| Readability | NL-s | 97.1 | 84.9 | 0.04 | 0.02 | 0.60 | 3066.5 | 0.06 |
| | NL-r | 96.8 | 84.8 | 0.02 | 0.02 | 0.55 | 3126.1 | 0.11 |
| | St | 96.6 | 84.4 | 0.03 | 0.02 | 0.56 | 3091.1 | 0.13 |

**Table 2: Primary paired comparison vs. NL-simple baseline per problem (Wilcoxon signed-rank, Cliff's $\delta$, Holm-corrected $p$ within each comparison). $\delta > 0$ favors the intervention (NL-rich or Structured) over NL-simple.**

| NFR | Comp. | Metric | $n$ | $p$ | $p_{\text{Holm}}$ | $\delta$ |
|---|---|---|---|---|---|---|
| Performance | NL-r vs NL-s | Pass@1 | 164 | 0.263 | 0.527 | 0.046 (neg.) |
| | | ET-Pass@1 | 164 | 0.843 | 0.843 | 0.022 (neg.) |
| | | Time | 164 | 0.000 | 0.000 | 1.000 (lg.) |
| | St vs NL-s | Pass@1 | 164 | 0.922 | 0.922 | 0.022 (neg.) |
| | | ET-Pass@1 | 164 | 0.182 | 0.364 | 0.002 (neg.) |
| | | Time | 164 | 0.000 | 0.000 | 1.000 (lg.) |
| Error Handling | NL-r vs NL-s | Pass@1 | 164 | 0.025 | 0.025 | 0.004 (neg.) |
| | | ET-Pass@1 | 164 | 0.001 | 0.001 | -0.027 (neg.) |
| | | Time | 164 | 0.000 | 0.000 | -1.000 (lg.) |
| | St vs NL-s | Pass@1 | 164 | 0.154 | 0.154 | 0.007 (neg.) |
| | | ET-Pass@1 | 164 | 0.024 | 0.048 | -0.004 (neg.) |
| | | Time | 164 | 0.000 | 0.000 | -0.999 (lg.) |
| Code Smell | NL-r vs NL-s | Pass@1 | 164 | 0.536 | 1.000 | 0.033 (neg.) |
| | | ET-Pass@1 | 164 | 0.656 | 1.000 | 0.015 (neg.) |
| | | Time | 164 | 0.000 | 0.000 | 0.988 (lg.) |
| | St vs NL-s | Pass@1 | 164 | 0.776 | 1.000 | 0.023 (neg.) |
| | | ET-Pass@1 | 164 | 0.600 | 1.000 | 0.010 (neg.) |
| | | Time | 164 | 0.000 | 0.000 | 0.988 (lg.) |
| Readability | NL-r vs NL-s | Pass@1 | 164 | 0.657 | 1.000 | 0.022 (neg.) |
| | | ET-Pass@1 | 164 | 1.000 | 1.000 | 0.024 (neg.) |
| | | Time | 164 | 0.000 | 0.000 | 0.988 (lg.) |
| | St vs NL-s | Pass@1 | 164 | 0.898 | 1.000 | 0.029 (neg.) |
| | | ET-Pass@1 | 164 | 0.524 | 1.000 | 0.019 (neg.) |
| | | Time | 164 | 0.000 | 0.000 | 0.988 (lg.) |

$\delta > 0$: intervention better than NL-simple baseline.

for Error Handling ET-Pass@1). Quality densities at the aggregate level differ only slightly between forms (e.g., unreadability Performance 0.69 vs. 0.79). Form can affect execution time selectively (large $\delta$ for Code Smell and Readability) but, as above, we treat time cautiously.

## 5.5 Synthesis Across Research Questions

Reading RQ1–RQ4 together yields a coherent pattern rather than four disconnected null results. **RQ1** (correctness vs. baseline) is largely null except for Error Handling, where richer fault-tolerance text appears to trade extended-test pass rate for explicit exception structure. **RQ2** compares NFR-specific quality against NL-simple (Table 3). Unreadability density improves for all four NFRs. Code-smell density improves when the prompt names that attribute, though the paired per-problem test for Code Smell is borderline after Holm correction. **RQ3** (robustness vs. baseline): condition-level Pass@1 STDEV reductions remain descriptive after Holm correction on per-problem tests, but per-problem unreadability

**Table 3: Paired per-problem comparison of quality densities vs. NL-simple baseline (Wilcoxon signed-rank, Cliff's $\delta$, Holm-corrected $p$ within each comparison). For density metrics, lower is better: $\delta < 0$ favors the intervention.**

| NFR | Comp. | Metric | $n$ | $p$ | $p_{\text{Holm}}$ | $\delta$ |
|---|---|---|---|---|---|---|
| Performance | NL-r vs NL-s | Unread. | 164 | 0.000 | 0.000 | -0.224 (sm.) |
| | | Smell | 164 | 0.893 | 0.893 | -0.021 (neg.) |
| | | Exc. | 164 | 0.002 | 0.003 | 0.027 (neg.) |
| | St vs NL-s | Unread. | 164 | 0.000 | 0.000 | -0.101 (neg.) |
| | | Smell | 164 | 0.943 | 0.943 | -0.029 (neg.) |
| | | Exc. | 164 | 0.001 | 0.002 | 0.015 (neg.) |
| Error Handling | NL-r vs NL-s | Unread. | 164 | 0.000 | 0.000 | -0.398 (med.) |
| | | Smell | 164 | 0.000 | 0.000 | -0.184 (sm.) |
| | | Exc. | 164 | 0.000 | 0.000 | 0.270 (sm.) |
| | St vs NL-s | Unread. | 164 | 0.000 | 0.000 | -0.354 (med.) |
| | | Smell | 164 | 0.000 | 0.000 | -0.179 (sm.) |
| | | Exc. | 164 | 0.000 | 0.000 | 0.239 (sm.) |
| Code Smell | NL-r vs NL-s | Unread. | 164 | 0.000 | 0.000 | -0.343 (med.) |
| | | Smell | 164 | 0.049 | 0.099 | -0.048 (neg.) |
| | | Exc. | 164 | 0.716 | 0.716 | -0.019 (neg.) |
| | St vs NL-s | Unread. | 164 | 0.000 | 0.000 | -0.236 (sm.) |
| | | Smell | 164 | 0.237 | 0.473 | -0.047 (neg.) |
| | | Exc. | 164 | 0.894 | 0.894 | -0.047 (neg.) |
| Readability | NL-r vs NL-s | Unread. | 164 | 0.000 | 0.000 | -0.163 (sm.) |
| | | Smell | 164 | 0.408 | 0.408 | -0.011 (neg.) |
| | | Exc. | 164 | 0.000 | 0.001 | -0.055 (neg.) |
| | St vs NL-s | Unread. | 164 | 0.000 | 0.000 | -0.132 (neg.) |
| | | Smell | 164 | 0.587 | 0.587 | -0.022 (neg.) |
| | | Exc. | 164 | 0.001 | 0.003 | -0.049 (neg.) |

$\delta < 0$: intervention lower density than NL-simple (improvement for smell/unreadability). For exception density, higher values in the intervention are consistent with the Error Handling NFR's fault-tolerance intent; see Section 5.2 for interpretation.

**Table 4: Secondary paired comparison *Structured vs. NL-rich* (form only; content held constant). $\delta > 0$ favors Structured.**

| NFR | Comp. | Metric | $n$ | $p$ | $p_{\text{Holm}}$ | $\delta$ |
|---|---|---|---|---|---|---|
| Performance | St vs NL-r | Pass@1 | 164 | 0.162 | 0.485 | -0.023 (neg.) |
| | | ET-Pass@1 | 164 | 0.477 | 0.727 | -0.019 (neg.) |
| | | Time | 164 | 0.363 | 0.727 | 0.125 (neg.) |
| Error Handling | St vs NL-r | Pass@1 | 164 | 0.149 | 0.149 | 0.003 (neg.) |
| | | ET-Pass@1 | 164 | 0.040 | 0.080 | 0.021 (neg.) |
| | | Time | 164 | 0.001 | 0.002 | 0.164 (sm.) |
| Code Smell | St vs NL-r | Pass@1 | 164 | 0.458 | 0.915 | -0.011 (neg.) |
| | | ET-Pass@1 | 164 | 0.495 | 0.915 | -0.006 (neg.) |
| | | Time | 164 | 0.000 | 0.000 | -0.794 (lg.) |
| Readability | St vs NL-r | Pass@1 | 164 | 0.833 | 1.000 | 0.006 (neg.) |
| | | ET-Pass@1 | 164 | 0.551 | 1.000 | -0.005 (neg.) |
| | | Time | 164 | 0.000 | 0.000 | 0.964 (lg.) |

$\delta > 0$: Structured better than NL-rich.

STDEV is significantly lower for enriched prompts in all four NFRs ($p_{\text{Holm}} < 0.05$), despite shorter and less lexically diverse prompt text (Table 5). **RQ4** (form) is null for correctness: once ISO content is fixed, JSON and prose are empirically interchangeable for Pass@1 and ET-Pass@1 on this model and benchmark.

The "so what" is therefore not "Structured beats NL-rich" or vice versa, but "ISO-grounded enrichment changes *quality proxies* and *stability* relative to one-line NFRs without buying functional correctness, and format is a secondary engineering choice." This ordering matches how practitioners should prioritize effort: invest in standard-grounded *content* first; choose JSON or prose based on tooling, not expected LLM performance gains.

## 6 Discussion

Our results support a single narrative: *ISO-grounded enrichment improves quality proxies and robustness relative to terse one-line NFRs, but does not reliably improve functional correctness; when content is fixed, representation form is secondary.* This section unpacks that claim, relates it to RobuNFR, and draws implications for researchers and practitioners in component-based and quality-aware LLM workflows.

**Table 5: Robustness (STDEV across ten prompt variations; lower = more stable) and prompt diversity (mean pairwise Jaccard distance). Condition-level STDEV is descriptive; per-problem STDEV inferential tests are in Section 4.8 and the replication package (Section 8).**

| NFR | Cond. | Pass@1 | Exc. | Smell | Unread. | Div. |
|---|---|---|---|---|---|---|
| Performance | NL-s | 0.0158 | 0.010 | 0.006 | 0.123 | 0.584 |
| | NL-r | 0.0057 | 0.006 | 0.007 | 0.037 | 0.123 |
| | St | 0.0109 | 0.004 | 0.008 | 0.037 | 0.101 |
| Error Handling | NL-s | 0.0151 | 0.361 | 0.009 | 0.172 | 0.595 |
| | NL-r | 0.0092 | 0.083 | 0.003 | 0.015 | 0.134 |
| | St | 0.0087 | 0.100 | 0.004 | 0.017 | 0.109 |
| Code Smell | NL-s | 0.0076 | 0.042 | 0.008 | 0.096 | 0.437 |
| | NL-r | 0.0129 | 0.014 | 0.003 | 0.021 | 0.107 |
| | St | 0.0096 | 0.011 | 0.004 | 0.021 | 0.088 |
| Readability | NL-s | 0.0050 | 0.013 | 0.009 | 0.102 | 0.549 |
| | NL-r | 0.0071 | 0.004 | 0.005 | 0.011 | 0.127 |
| | St | 0.0077 | 0.007 | 0.005 | 0.020 | 0.102 |

## 6.1 Enrichment Matters; Correctness Does Not Follow Automatically

The primary comparison against NL-simple shows a deliberate trade-off rather than uniform failure or success. ISO-grounded paragraphs and JSON objects add constraints, acceptance criteria, and explicit mappings to ISO/IEC 25010 characteristics. Those additions give the model a narrower, more semantically anchored interpretation of the NFR. That anchoring appears to reduce Pylint Convention issues (unreadability density) and, for the Code Smell NFR, Refactor warnings, without systematically raising Pass@1 or ET-Pass@1.

For three of four NFRs (Performance, Code Smell, Readability), correctness effects are negligible ($|\delta| \leq 0.033$ after Holm). This suggests that enrichment neither "fixes" nor "breaks" functional synthesis on HumanEval for maintainability- and performance-oriented phrasing, at least under greedy decoding with a strong contemporary model. Practitioners should not expect ISO-grounded NFR paragraphs to substitute for tests or to raise benchmark pass rates; the benefit lies elsewhere (Section 6.2).

Error Handling is the exception. Richer fault-tolerance specifications significantly reduce ET-Pass@1 ($p_{Holm} = 0.001$ for NL-rich; 0.048 for Structured) while *increasing* exception density. We stress that the rise in exception density is **ambiguous evidence**: it cannot be read as a clean reliability improvement, because the same added `try`/`except` and validation branches that satisfy the NFR text are the most plausible cause of the ET-Pass@1 drop on HumanEval's exact-output oracles. In other words, the very signal that looks like NFR satisfaction in a static view (more explicit error handling) is entangled with the functional regression in a dynamic view. Disentangling "better defensive design" from "benchmark incompatibility" would require oracles that reward fault handling, which HumanEval does not provide (Section 6.6). This aligns with RobuNFR's finding that NFR-aware prompts can reduce Pass@1 [10]: here, the reduction is tied to *more explicit* reliability requirements rather than to surface re-wording alone. For teams adopting ISO-grounded error-handling NFRs, we recommend validating against extended tests and domain oracles, not assuming benchmark pass rate as a proxy for reliability satisfaction.

## 6.2 Quality and Robustness Gains

Static-analysis densities improve in a direction consistent with the NFR intent. Unreadability density falls for every NFR (e.g., Performance NL-simple 0.88 vs. NL-rich 0.69, a 22% relative drop at aggregate level). Code-smell density drops when the NFR targets that attribute (0.019 → 0.011 for NL-rich). These are proxy metrics, not user-perceived quality, but they operationalize maintainability-related NFRs in a way comparable to RobuNFR and prior ISO-oriented work [14].

Robustness: Table 5 shows lower condition-level Pass@1 STDEV for Performance and Error Handling under enrichment. Per-problem Wilcoxon tests do not confirm Pass@1 STDEV gains after Holm correction, but *do* confirm lower unreadability STDEV for every intervention–baseline pair ($p_{\text{Holm}} < 0.05$). Intervention prompts are *shorter* and have *lower* mean pairwise Jaccard distance than NL-simple ($\approx$ 0.10–0.13 vs. 0.44–0.59); part of that gap reflects shared ISO template boilerplate. Lower unreadability STDEV is therefore not explained by more varied prompt text alone; enrichment constrains static-quality outcomes under re-wording even when functional Pass@1 variation remains descriptive.

## 6.3 Form Is Secondary When Content Is Fixed

RQ4 asks whether LLMs "prefer" JSON over prose when ISO fields are identical. The answer on correctness is no: all Pass@1 and ET-Pass@1 contrasts have negligible Cliff's $\delta$ and fail to reach significance after Holm correction. Aggregate quality densities differ only slightly (e.g., unreadability Performance 0.69 vs. 0.79). Teams integrating NFRs from architecture models, quality portals, or MDE pipelines can serialize requirements as JSON for traceability and tooling without expecting a correctness penalty relative to the same content in prose *on this model*. We do not claim universal null effects for all models or tasks; we claim that format was not the dominant lever in our controlled setting.

## 6.4 Per-NFR Patterns

**Performance.** Enrichment does not move Pass@1/ET-Pass@1 but lowers unreadability density; Pass@1 STDEV reductions in Table 5 are descriptive after Holm correction on per-problem tests, while unreadability STDEV gains are inferentially supported (Section 5.3). Mean execution time rises vs. NL-simple (large positive $\delta$); we treat timing as indicative only (Section 7).

**Error Handling.** Enrichment increases exception density and reduces ET-Pass@1, while also lowering unreadability density and prompt sensitivity. The NFR is partially satisfied in static structure but conflicts with benchmark tests on a subset of problems.

**Code Smell.** Correctness remains stable; code-smell and unreadability densities improve. This is the clearest alignment between NFR intent and proxy metric movement.

**Readability.** Similar to Code Smell for correctness and unreadability; smell density was already low at baseline, leaving less headroom for improvement.

## 6.5 Relation to RobuNFR and the "One-Line" Baseline

Lin et al. [10] establish that NFR-aware generation is sensitive to prompt variation and can reduce Pass@1. Our NL-simple baseline reproduces that sensitivity pattern: higher Pass@1 STDEV and higher lexical diversity than enriched prompts. RobuNFR does not ask whether *enriching* the NFR content (beyond re-wording the same terse phrase) changes quality and robustness; we show that it does for static-analysis proxies and stability, even when correctness is flat or slightly worse. Comparing interventions only against each other, without a one-line baseline, would have masked this: RQ4's null form result does *not* imply enrichment is useless (Section 6.7).

## 6.6 What HumanEval Can and Cannot Tell Us About NFRs

Our conclusions are bounded by what the benchmark can observe, and this boundary is itself part of the finding rather than a footnote. HumanEval problems are small, self-contained Python functions with exact-output unit tests. This design is well suited to *functional* correctness but only weakly sensitive to most ISO/IEC 25010 quality characteristics. Three consequences follow. First, the NFRs we can measure here are necessarily the ones with *intra-function, statically observable* signatures (readability/code-smell via Pylint, local exception structure, execution time of a single call); genuinely architectural concerns such as modifiability across modules, compatibility, scalability, or security are out of reach on single-function tasks. Second, the benchmark's oracles actively penalize some legitimate NFR behavior: a function that validates inputs and raises specific exceptions can be "more reliable" in the ISO sense yet fail an exact-output test, which is exactly the Error Handling tension we observe. The metric and the oracle thus disagree by construction, not by accident. Third, density proxies are most meaningful as *relative* within-task contrasts (intervention vs. baseline on the same problem), which is why we pair per problem; they should not be read as absolute maintainability scores.

This framing tempers our claims in a specific way. The RQ2 quality gains and RQ3 stability gains are real and significant within HumanEval's observable window, but they speak to *function-level* static quality and prompt sensitivity, not to system-level maintainability or reliability. Likewise, the RQ4 null form effect holds for short functions; structured specifications might matter more when NFRs constrain interfaces, configuration, or cross-component contracts that a single-function benchmark cannot express. We therefore read our results as a lower bound on where representation could matter: if even content-rich enrichment leaves correctness flat on small functions, the leverage of NFR specification is more likely to appear in repository- and architecture-scale generation, which we flag as the priority for future benchmarks (Section 7).

## 6.7 Implications

**For researchers.** (i) Report enrichment level explicitly when studying NFR-aware LLM generation; a null form-only comparison is insufficient to characterize ISO-grounded interventions. (ii) Pair functional benchmarks with NFR-aligned proxy metrics and extended tests; Error Handling shows they can diverge. (iii) Measure prompt sensitivity alongside correctness; robustness is an outcome variable in its own right for NFR studies.

**For practitioners.** (i) Adopt ISO/IEC 25010-grounded NFR text when the goal is cleaner, more stable generated code along maintainability dimensions, not higher HumanEval pass rates. (ii) Choose JSON vs. prose based on pipeline integration, not expected LLM gains. (iii) For reliability NFRs, review generated exception structure against project oracles; richer specs may reduce benchmark compatibility while increasing explicit fault-handling code. (iv) **Cost of specification:** ISO objects require upfront authoring (intent, constraints, acceptance criteria mapped to ISO/IEC 25010:2023). That effort pays off when teams already maintain quality models, architecture portals, or MDE pipelines that export structured attributes; for one-off scripts, a one-line NFR may suffice unless static-quality stability matters (an engineering-effort trade-off, not an API-cost claim).

**For CBSoft/SBCARS reuse contexts.** Component generators and architecture-centric workflows often maintain structured quality attributes alongside functional interfaces. Our results suggest exporting those attributes into LLM prompts is worthwhile for quality and stability, provided teams validate functional behavior independently.

## 6.8 Practical Decision Guide

Table 1 and the paired tests support a simple decision guide. If the goal is *maximize HumanEval pass rate*, NL-simple and enriched prompts perform similarly for three NFRs, and enriched Error Handling may hurt ET-Pass@1; a one-line NFR is not clearly worse and may avoid spec-test conflicts. If the goal is *reduce Pylint Convention/Refactor issues* or *stabilize outputs across re-wordings*, ISO-grounded enrichment is preferable regardless of JSON vs. prose. If the goal is *tool integration*, Structured JSON is supported without correctness penalty relative to NL-rich in RQ4. These recommendations are conditional on `gpt-5.4-2026-03-05`and HumanEval; they illustrate how the primary/secondary RQ ordering translates into engineering choices rather than treating format comparison as the paper's main contribution.

# 7 Threats to Validity

**Construct validity.** NFR-quality is operationalized through static-analysis proxies (Pylint Refactor/Convention densities, exception statements) and execution time rather than human judgment. These proxies align with RobuNFR [10] and ISO-oriented LLM quality work [14] but do not capture runtime reliability, security, or user-perceived maintainability. **Exception-statement density** is an ambiguous proxy for reliability: models may add broad or low-quality `try`/`except` blocks without improving fault tolerance (Section 5.2). Mean pairwise Jaccard distance mixes surface variation with fixed ISO template text; lower diversity on enriched prompts should not be read as semantic equivalence alone. The NFR→ISO/IEC 25010 mapping is a researcher decision; alternative mappings (e.g., placing error handling under Maintainability) could change prompt emphasis. We make the mapping explicit and traceable in the replication package. Acceptance criteria never instruct the model to pass tests, and the functional task clause is identical across conditions. For error handling, Pass@1 can penalize spec-conflicting exceptions,

so we additionally rely on exception density and ET-Pass@1 and discuss the trade-off in Section 6.

**Internal validity.** Code-quality metrics are normalized per ten LOC; correctness comparisons are paired per problem. Decoding is greedy (temperature 0). A central threat is **non-concurrent collection**: the NL-simple baseline predates NL-rich and Structured by several weeks (Apr–May vs. June 2026). We fixed the same model snapshot (`gpt-5.4-2026-03-05`), temperature 0, benchmark, and evaluation pipeline across conditions to limit API drift, and OpenAI documents that snapshots lock model versions [13]. We re-ran NL-simple prompt0 as a stability control (Section 4.6): on all 164 Performance tasks (August 2026), Pass@1 was 0.93 vs. 0.92 ($p = 0.424$; bootstrap 95% CI $[-0.04, +0.01]$, including zero); a prior June pilot on 30 stratified tasks showed a larger apparent shift that we attribute to subset bias ($n = 30$, Performance $p = 0.072$). We therefore treat snapshot drift on Performance as unlikely to dominate the primary comparisons, while noting that the baseline and interventions were still not collected concurrently and that Error Handling was assessed only on the small pilot. Residual batch effects unrelated to the model snapshot (e.g., infrastructure changes) cannot be ruled out; the replication package documents pipeline versions, hosts all raw outputs, and includes W1 stability JSONs (`w1_stability_comparison.json`, `w1b_stability_performance_164.json`) in `results/w1-stability`.

**Conclusion validity.** We use non-parametric paired Wilcoxon signed-rank tests [16], report effect sizes (Cliff's delta), and correct for multiple comparisons (Holm–Bonferroni) separately within each (NFR, comparison, metric family). Correctness, time, and density metrics all use per-problem paired vectors ($n = 164$). For density metrics where lower is better, $\delta < 0$ favors the intervention. Significance with negligible $\delta$ (Error Handling ET-Pass@1; Code Smell paired smell density) is interpreted as a small but systematic shift across many problems, not a large practical swing.

**External validity.** One model (`gpt-5.4-2026-03-05`), one benchmark (HumanEval/ET), four NFRs; findings are tied to this snapshot and may not generalize to repository-level generation, multi-file tasks, or weaker models. As discussed in Section 6.6, HumanEval's small single-function tasks and exact-output oracles make it only weakly sensitive to system-level ISO/IEC 25010 characteristics, so our quality and form findings are bounded to function-level static quality and prompt sensitivity. We position the work as extending RobuNFR's prompt-variation methodology with ISO-grounded enrichment and a secondary form factor; MBPP and HumanEval-NFR replication, and repository-/architecture-scale NFR benchmarks, are listed as future work.

**Reliability/Reproducibility.** Execution time depends on hardware; we compare only internal deltas within the same experimental run. The full pipeline and data are released in the replication package (Section 8), including `results_numbers.json` traceability from tables to source files.

## 8 Conclusion and Future Work

We compared two ISO/IEC 25010:2023-grounded NFR interventions (NL-rich prose and Structured JSON) against the RobuNFR-style NL-simple baseline for four NFRs on HumanEval/ET with `gpt-5.4-2026-03-05`. **Primary result:** ISO-grounded enrichment improves static-analysis quality densities and reduces prompt sensitivity versus one-line NFRs, but does not reliably improve functional correctness and, for Error Handling, may reduce extended-test pass rate. **Secondary result:** when content is held constant, prose versus JSON form has negligible effect on correctness. In one sentence: *what* you specify in an NFR (standard-grounded enrichment) matters more for quality and robustness than *how* you serialize it (NL-rich vs. JSON), while functional correctness remains dominated by the functional task and benchmark oracles.

Future work includes MBPP replication, additional models and decoding settings beyond `gpt-5.4-2026-03-05`, human evaluation of ISO-aligned quality, and direct measurement of requirement ambiguity in NL-simple vs. enriched prompts. Extending the study to architecture-centric benchmarks (e.g., HumanEval-NFR [6]) would test whether enrichment benefits transfer when functional and non-functional requirements are co-specified.

## Artifact Availability

A replication package (prompt configurations, generated completions, evaluation outputs, analysis scripts, documentation; MIT License) is available at `https://doi.org/10.5281/zenodo.22135439` (v1.0.6; `LICENSE` and `README.md` at the archive root; camera-ready PDF at `docs/sbcars2026-camera-ready.pdf`). The package includes `results_numbers.json` (table and $p$-value traceability) and `results/rq3_per_problem_stdev.json` (RQ3 per-problem STDEV inferential tests). W1 snapshot-stability checks are in `w1_stability_comparison.json` (pilot, $n = 30$) and `w1b_stability_performance_164.json` (Performance, $n = 164$), under `results/w1-stability`.

## Acknowledgements

For partially supporting this work, we would like to thank INES.IA (National Institute of Science and Technology for Software Engineering Based on and for Artificial Intelligence) www.ines.org.br, CNPq grant 408817/2024-0. This study was financed in part by the Coordenação de Aperfeiçoamento de Pessoal de Nível Superior - Brasil (CAPES) – Finance Code 001.

## A ISO Prompt Object Schema (Replication)

Each NL-rich and Structured prompt is generated from one JSON object per (NFR, variation). Keys are identical across forms; only serialization differs. Core fields:

- `attribute`: NFR identifier (`performance`, `error_handling`, `code_smell`, `readability`).
- `intent`: one-sentence goal aligned with ISO/IEC 25010 subcharacteristics.
- `iso_25010`: nested `characteristic` and `sub_characteristics` strings.
- `constraints`: list of imperative rules (typically three or four) scoped to the quality attribute.
- `acceptance_criteria`: list of checkable conditions; never reference HumanEval tests.

Variations alter lexical choice and clause order in `intent` and constraints while keeping `iso_25010` and acceptance criteria fixed. NL-simple prompts use RobuNFR-style one-line templates in `approach/` (`nfr_prompts.py`).

## References


[1] Jomar Thomas Almonte, Santhosh Anitha Boominathan, and Nathalia Nascimento. 2025. Automated Non-Functional Requirements Generation in Software Engineering with Large Language Models: A Comparative Study. arXiv:2503.15248 [cs.SE] https://arxiv.org/abs/2503.15248

[2] Jacob Austin, Augustus Odena, Maxwell Nye, Maarten Bosma, Henryk Michalewski, David Dohan, Ellen Jiang, Carrie Cai, Michael Terry, Quoc Le, and Charles Sutton. 2021. Program Synthesis with Large Language Models. *arXiv preprint arXiv:2108.07732* (2021). https://arxiv.org/abs/2108.07732

[3] Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, et al. 2021. Evaluating Large Language Models Trained on Code. *arXiv preprint arXiv:2107.03374* (2021). https://arxiv.org/abs/2107.03374

[4] Norman Cliff. 1993. Dominance Statistics: Ordinal Analyses to Answer Ordinal Questions. *Psychological Bulletin* 114, 3 (1993), 494–509. doi:10.1037/0033-2909.114.3.494

[5] Julian Frattini, Davide Fucci, Richard Torkar, Lloyd Montgomery, Michael Unterkalmsteiner, Jannik Fischbach, and Daniel Mendez. 2024. Applying Bayesian Data Analysis for Causal Inference about Requirements Quality: A Controlled Experiment. *Empirical Software Engineering* 29 (2024). doi:10.1007/s10664-024-10582-1

[6] Hojae Han, Jaejin Kim, Jaeseok Yoo, Youngwon Lee, and Seung-won Hwang. 2024. ArchCode: Incorporating Software Requirements in Code Generation with Large Language Models. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (ACL).* https://arxiv.org/abs/2408.00994

[7] Sture Holm. 1979. A Simple Sequentially Rejective Multiple Test Procedure. *Scandinavian Journal of Statistics* 6, 2 (1979), 65–70.

[8] International Organization for Standardization. 2023. *ISO/IEC 25010:2023 Systems and software engineering: Systems and software Quality Requirements and Evaluation (SQuaRE): Product quality model.* Technical Report. ISO/IEC. https://www.iso.org/standard/78176.html

[9] International Organization for Standardization. 2024. *ISO/IEC 25002:2024 Systems and software engineering: Systems and software Quality Requirements and Evaluation (SQuaRE): Quality model overview and usage.* Technical Report. ISO/IEC. https://www.iso.org/standard/78175.html

[10] Feng Lin, Dong Jae Kim, Zhenhao Li, Jinqiu Yang, and Tse-Hsun Chen. 2025. RobuNFR: Evaluating the Robustness of Large Language Models on Non-Functional Requirements Aware Code Generation. *arXiv preprint arXiv:2503.22851* (2025). doi:10.48550/arXiv.2503.22851

[11] Jiawei Liu, Chunqiu Steven Xia, Yuyao Wang, and Lingming Zhang. 2023. Is Your Code Generated by ChatGPT Really Correct? Rigorous Evaluation of Large Language Models for Code Generation. In *Advances in Neural Information Processing Systems (NeurIPS).* https://arxiv.org/abs/2305.01210

[12] Logilab and Pylint contributors. [n. d.]. Pylint: A Static Code Analyser for Python. https://pylint.readthedocs.io. Accessed June 2026.

[13] OpenAI. [n. d.]. GPT-5 Model. https://web.archive.org/web/20260611010413/https://developers.openai.com/api/docs/models/gpt-5. Snapshots documentation. Archived 2026-06-11 via the Internet Archive Wayback Machine; original at https://developers.openai.com/api/docs/models/gpt-5 (accessed June 2026).

[14] Xin Sun, Daniel Ståhl, Kristian Sandahl, and Christoph Kessler. 2025. Quality Assurance of LLM-generated Code: Addressing Non-Functional Quality Characteristics. *arXiv preprint arXiv:2511.10271* (2025). https://arxiv.org/abs/2511.10271

[15] Luiz Viviani, Eduardo Guerra, Jorge Melegati, and Xiaofeng Wang. 2023. An Empirical Study About the Instability and Uncertainty of Non-Functional Requirements. In *Agile Processes in Software Engineering and Extreme Programming (XP 2023) (Lecture Notes in Business Information Processing, Vol. 475).* Springer, 77–93. doi:10.1007/978-3-031-33976-9_6

[16] Frank Wilcoxon. 1945. Individual Comparisons by Ranking Methods. *Biometrics Bulletin* 1, 6 (1945), 80–83. doi:10.2307/3001968