
LLM-Assisted Ambiguity Detection in Regulatory Specifications

A GST Compliance Case Study

Kushagra Sharan

Manipal Institute of Technology, Manipal

kushagra.mitmpl2024@learner.manipal.edu

Secure Program Synthesis Hackathon · Apart Research · May 2026

Abstract

Regulatory software often begins with natural-language requirements that are understandable to humans but insufficiently precise for reliable implementation. This project studies that problem using GST-style compliance rules involving filing dates, invoices, actors, penalties, and exceptions. A lightweight Python pipeline was developed using three sequential LLM calls: requirement formalization, ambiguity auditing, and structured scoring. Using Google Gemini on seven GST requirements, the pipeline identified recurring underspecification patterns including missing temporal boundaries, undefined evidence requirements, unclear actor responsibilities, and incomplete exception handling.

The primary contribution is practical: an LLM-assisted review pass can help surface unresolved implementation questions before formal methods or software engineering workflows begin. The system is not intended as legal authority or proof infrastructure, but as a specification elicitation aid for regulatory software development.

Keywords: LLMs, specification engineering, regulatory compliance, ambiguity detection, formal methods, secure synthesis

1. Introduction

Formal verification and secure program synthesis depend on good specifications. In practice, many systems do not start with precise specifications; they start with policy text, product requirements, regulations, or informal domain-expert explanations. Dodds makes a similar point in “*Specifications Don’t Exist*” [1]: for many useful systems, the hard part is not only proving code correct, but deciding what the correct behavior should be.

This project applies that problem to GST/tax compliance rules. GST requirements are written for accountants, lawyers, and businesses, not for software engineers. A rule such as “*late filing attracts a penalty of Rs. 50 per day*” sounds precise, but software still needs to know when the clock starts, whether there is a cap, which return is covered, and what happens in exceptional cases.

The goal of this project is to build a lightweight **ambiguity finder** for regulatory requirements. It does **not** attempt to determine the correct legal answer. Instead, it identifies missing

specification (*spec*) details *before* implementation.

Our main contributions are:

1. A simple three-call LLM pipeline for *formalizing*, *attacking*, and *scoring* natural-language GST requirements.
2. A GST-focused case study showing repeated ambiguity patterns in tax compliance rules.
3. A reproducible Python script that saves structured JSON results for use by the general public.

2. Related Work

This project is closest to work on specification quality and AI-assisted formal methods. Dodds' work [1] suggests the idea that the specification itself is often the bottleneck. The Vericoding benchmark [2] studies the LLM generation of formally verified programs from *formal* specifications, which come later in the chain. This project focuses on whether the original human requirement is specific enough *before* formalization.

Atlas Computing's work on AI-assisted specification tools is also relevant [3]. Their specification IDE explores how natural language can be mapped to formal specifications and reviewed by humans. This project is more domain-specific: it does not build an IDE or generate proof artifacts. It focuses on a practical first-pass check for regulatory software requirements.

The gap addressed here is the step between plain-English regulation and implementation: **finding the missing questions** that developers should answer before writing code.

3. Methods

The input dataset is `requirements.txt`, with one GST requirement per line. The script `run_pipeline.py` reads each line and runs a three-call LLM pipeline, illustrated in Figure 1.



Figure 1. The three-stage pipeline. Each GST requirement is first formalized into explicit bullet-point rules, then audited for edge cases, and finally scored as structured JSON. All calls use fixed system prompts.

The three calls use fixed system prompts (full text in Appendix).

Call 1: Formalize. The model rewrites the requirement as explicit bullet-point specification rules, exposing implied conditions.

Call 2: Attack. The model audits the formalized spec and lists edge cases, ambiguities, and underspecified conditions as concise bullets.

Call 3: Score. The model returns strict JSON:

```

1 {
2   "ambiguity_score": 1,           // 1-10; higher = more underspecified
3   "primary_issue": "one sentence",
4   "verdict": "underspecified | overspecified | adequate"

```

5 }

Score bands: **1–3** mostly clear; **4–6** partly underspecified; **7–8** significantly underspecified; **9–10** highly underspecified.

The implementation uses Python’s standard library. Google AI Studio / Gemini is the primary API path; any OpenAI-compatible endpoint is supported as a fallback via the `-base-url` argument, with OpenRouter as the default.

4. Results

Table 1 shows the Gemini-pipelined results for all seven completed requirements.

Table 1. GST requirements scored by the ambiguity finder. All seven completed requirements were rated *underspecified*. Score range 1–10; higher = more underspecified.

Requirement	Score	Verdict	Primary Issue
Input tax credit claimable only if supplier has filed return	9	underspecified	Definition of “ <i>filed</i> ,” timing of verification, and handling of post-filing scenarios are all missing.
Late filing of GST return attracts penalty of Rs. 50 per day per return	9	underspecified	Start, end, and scope of the penalty period are undefined.
Nil return must be filed even if no transactions in a period	9	underspecified	“Period” and “transaction” are undefined; no operational filing context provided.
Input tax credit cannot be claimed on goods for personal consumption	9	underspecified	“Personal consumption” and “goods” scope both undefined; no operational determination rules.
Invoice must be issued within 30 days of supply of service	8	underspecified	Exact start and end dates of the 30-day window relative to supply date are not specified.
Composition dealer cannot collect GST from customers	7	underspecified	Definition of “collecting,” scope of supplies, and consequences of erroneous collection are unclear.
Exports are zero-rated and eligible for refund of input tax credit	8	underspecified	Definition of “export,” refund criteria and process, and applicable exceptions are all absent.

Every completed requirement was marked underspecified. Concise natural-language regulatory rules leave out substantial implementation detail. The most common ambiguity types are visualised in Figure 2.

5. Discussion and Limitations

The results confirm that regulatory language has systematic ambiguity patterns. The project is useful because these are exactly the questions developers face when converting compliance rules into software. An LLM pipeline can act as a **simple and cheap first-pass reviewer** before implementation begins.

In a tax compliance system, an ambiguous rule is not just a correctness gap; a compliant-looking implementation of an incomplete spec can allow fraudulent credit claims to pass undetected.



Figure 2. The four recurring ambiguity categories found across all seven requirements. Circled numbers indicate how many of the seven requirements exhibited each category.

Limitations

The main limitation is that model output is not ground truth. An LLM may flag false positives, miss real legal issues, or hallucinate plausible-sounding but fictitious edge cases. The project also uses a very limited sample of GST-style requirements rather than a labeled benchmark. Outputs should be read as *questions for review*, not as legal conclusions.

Future Work

The projected next steps are: (1) validate model-flagged ambiguities against expert-labeled examples or real tax disputes, and increase the sample size; (2) extend to other compliance domains such as KYC, payroll, insurance claims, or privacy policy enforcement; and (3) add human review, source citations to legal sections, and multi-model comparison.

6. Conclusion

This project built a small, reproducible ambiguity finder for GST-style regulatory requirements. The pipeline formalizes a requirement, attacks the formalized version, and scores the ambiguity as JSON. In the completed Gemini run, all seven analyzed requirements were rated underspecified, with repeated issues around time, actors, evidence, and exceptions.

The main lesson is simple: regulatory requirements often need spec elicitation before implementation. LLMs are not a substitute for legal review or formal proof, but they can help surface missing questions early enough to be useful.

Code and Data.

- **Code repository:** [GitHub](#)
- **Data:** `requirements.txt` contains the GST-style sample requirements.
- **Other artifacts:** `results/output.json`

Author Contributions. Kushagra Sharan designed the project, implemented the Python pipeline, ran the experiments, analyzed the results, and wrote the report.

References

- [1] Mike Dodds. “*Specifications Don’t Exist.*” Galois, 2025. <https://www.galois.com/articles/specifications-dont-exist>

- [2] Sergiu Bursuc et al. “A Benchmark for Vericoding: Formally Verified Program Synthesis.” arXiv:2509.22908, 2025. <https://arxiv.org/abs/2509.22908>
- [3] Evan Miyazono, Daniel Windham, and Alexandre Rademaker. “IDE for Validating Specifications.” Atlas Computing, 2025. <https://blog.atlascomputing.org/p/ide-for-validating-specifications>
- [4] Google. “Gemini API: Generate Content,” 2026. <https://ai.google.dev/api/generate-content>

Appendix

A. System Prompts

Formalize system prompt

```
1 You are a formal specification writer. Convert requirements to precise,  
2 unambiguous bullet-point specs. No prose. Every condition explicit.
```

Attack system prompt

```
1 You are a specification auditor. Return a concise bullet list of edge  
2 cases, ambiguities, and underspecified conditions. No intro. No headings.
```

Score system prompt

```
1 You are a requirements quality rater. Output JSON only.
```

B. Limitations and Dual-Use Considerations

False positives are possible: the model may flag issues that a tax expert considers obvious from statutory context. There is also no formal proof that the found ambiguities are real. The tool could be misused to create confusion around compliance rules, so outputs should be presented as questions for review rather than legal conclusions.

C. LLM Usage Statement

LLM assistance was used to help draft the project framing, code structure, and report formatting. The pipeline results in `results/output.json` were generated through Google Gemini API calls where quota allowed. All claims in the report were reviewed against the generated output and the local code.