Grp. 3-11: Godot Issue Triage with LLMs: Converting Natural-Language Bug Reports into Structured Routing Records

Gian Paul Ramirez
gi351142@ucf.edu
University of Central Florida
Orlando, Florida, USA

1 Problem Statement

Open-source projects receive large volumes of GitHub issues. This project focuses on issues for Godot, a popular open-source game engine. Issue reports vary widely in quality and structure: multiple concerns can be mixed, repro steps may be missing, and platform details are often incomplete. This makes manual triage slow and unreliable. Poor triage reduces engineering efficiency as misrouted or poorly summarized issues increase time-to-fix and delay high-impact bugs that affect both players and developers.

The goal is to automate routing and enforce structured triage records. **Given an issue's text, an LLM will generate a standardized triage record** to enable faster, more consistent assignment and prioritization with measurable reliability.

2 Proposed Technique

For this project, **10k** issues will be sampled from Godot's GitHub repository. For each issue the **title, body, labels, and metadata will be extracted**. To ensure consistent structured outputs and defensible evaluation, the following set vocabulary of labels is defined and mapped into a triage schema.

2.1 Primary Routing Targets (Components)

The project focuses on the subsystem/component labels with the `topic:*` prefix. The `components` field will **act as our target**, with routing framed as **multi-label prediction**. Concretely, the top N `topic:*` labels by frequency (with $N \in [15, 25]$) will be selected, a minimum support threshold will be enforced (e.g., at least 200 issues per label), and all remaining low-frequency topic:* labels will be mapped into an `other` category.

2.2 Extracted (Non-Routing) Fields and Excluded Labels

The following non-routing labels will be extracted into separate fields: `platform:*` (platform), issue type (`bug`/`enhancement`/...), and impact flags (`regression`/`crash`/`high priority`/...). Some labels will be excluded to reduce leakage and noise: workflow/status, maintenance/release, moderation, and meta/process labels (e.g., `confirmed`, `spam`, etc.).

2.3 LLM Interface

For each issue, the title and body will be input and an LLM query performed via API with a fixed prompt defining the set label vocabulary, label normalization rules, and the requirement to emit a *single* JSON object matching the provided schema. **We run N sampled generations and aggregate predictions across samples** (majority vote will be used for categorical fields and top-k/union for multi-label fields), with confidence defined as how consistently the model predicts the same value for each JSON field across the sampled generations. Outputs will be validated against required fields, enums, and bounds; malformed JSON will trigger repair/retry using the validation error, and low-confidence outputs will be flagged `needs_human_triage`, indicating abstention.

2.4 Schema structure

Labels will be normalized as: `enhancement`/`feature proposal` → `feature_request`, `documentation` → `docs`, and `topic:*`/`platform:*` values are mapped by prefix-stripping (e.g., `topic:gui` → `gui`). The LLM must output *one* JSON object conforming to the schema below:

```
{
  "issue_type": "bug|feature_request|discussion|
      docs|other",
  "components": ["rendering", "gui", ...],
  "platform": ["windows|linuxbsd|macos|android|web
      |unknown"],
  "impact": ["high priority", "regression", ...],
  "needs_human_triage": false,
  "confidence": 0.0
}
```

2.5 Evaluation

Evaluation uses a **time-based split** to simulate drift: tune on the oldest 80% of issues and test on the newest 20%. Multi-label routing quality will be reported via **top-k accuracy** and **micro/macro F1**, plus schema validity (first-pass and post-repair). The coverage–accuracy tradeoff will be quantified under abstention and test robustness by noising issue text and measuring routing stability.

3 Expected Outcomes

The aim is to deliver an end-to-end triage pipeline that ingests Godot GitHub issues and outputs validated, structured routing records. Success will be demonstrated by strong routing performance on the time-split test set, with schema compliance of at least 90% on first-pass generation and at least 95% after automated repair. We will also provide a coverage–accuracy tradeoff curve to characterize abstention behavior and identify a practical operating point for deployment.

4 References

- https://github.com/godotengine/godot
- https://webdocs.cs.ualberta.ca/ dale/papers/iclr23b.pdf
- https://aclanthology.org/2025.tacl-1.26/
- https://platform.openai.com/docs/guides/structured-outputs