**Now together ai code is the main target to improve, what are the improvements that could be made.**

Improving your current Together AI LLM cleaner script can significantly boost its **reliability**, **speed**, **cost-efficiency**, and **clean output quality**. Here's a comprehensive list of potential improvements, grouped by purpose:

**🛡️ 1. Robustness and Error Handling**

| **Area** | **Suggestion** | **Why It Matters** |
| --- | --- | --- |
| ✅ Retry Logic | Add upper retry limits, exponential backoff with jitter | Prevent infinite loops or simultaneous retries |
| ✅ JSON Parsing | Add fallback to regex extraction for malformed JSON | LLMs occasionally hallucinate or produce invalid JSON |
| ✅ Rate Limit Handling | Check for HTTP 429 and implement wait/retry | Prevents pipeline crashes from temporary API overloads |
| ✅ Timeout Settings | Set request timeouts (e.g., 30s) | Prevent long-hanging requests blocking the job |
| ✅ Output Validation | Ensure keys like is\_valid, problem, solution always exist | Prevents downstream ETL errors or schema issues |

**🚀 2. Performance & Cost Optimization**

| **Area** | **Suggestion** | **Why It Matters** |
| --- | --- | --- |
| 🔄 Batch Inference | Send multiple posts per API call (if Together AI supports it) | Reduces request overhead and API call cost |
| 📉 Model Selection | Consider smaller models for pre-filtering | Use cheaper models to skip irrelevant posts |
| 💤 Early Exit | If post is too short or obviously irrelevant, skip before LLM | Reduces API costs and speeds up processing |

**✨ 3. Output Quality Improvements**

| **Area** | **Suggestion** | **Why It Matters** |
| --- | --- | --- |
| 🧠 Prompt Engineering | Add more diverse examples (valid/invalid, long/short) | Helps the LLM generalize better and avoid hallucinations |
| 🧪 Test Prompt Variants | Test versions with clearer step-by-step reasoning prompts | May improve accuracy in edge cases |
| 🔍 Post-Validation Heuristics | If problem is too vague or solution < 5 words → flag for review | Filters out bad LLM completions from production |

**🧱 4. Code Architecture & Maintainability**

| **Area** | **Suggestion** | **Why It Matters** |
| --- | --- | --- |
| 📦 Modularization | Split into: prompt builder, LLM client, validator, writer | Improves readability and testability |
| ⚙️ Config System | Centralize all configs: model, retries, save paths, etc. | Avoid hardcoded values and enable easy tuning |
| 📈 Logging | Use logging instead of print, with levels (INFO, WARNING, ERROR) | Supports better debugging in production/GitHub Actions |
| ✅ Test Harness | Unit test for LLM output parsing and edge cases | Detects regressions early when refactoring |

**🗃️ 5. Integration Readiness**

| **Area** | **Suggestion** | **Why It Matters** |
| --- | --- | --- |
| 🧪 Dry Run Mode | Add CLI flag to run without calling the API | Helps debug GitHub workflows without wasting API calls |
| 📂 Artifact Storage | Save output, errors, logs as GitHub artifacts | Makes GitHub Action runs reproducible and auditable |
| 🔗 GitHub Caching | Cache dependencies and model loaders | Speeds up workflow runs |

**🧠 6. Advanced Features (Optional)**

| **Area** | **Suggestion** | **Why It Matters** |
| --- | --- | --- |
| 🧮 Confidence Scoring | Have the LLM add a confidence score to each answer | Useful for downstream filtering and UI display |
| 📊 Metrics Reporting | Count valid/invalid, average token usage, time per call | Helps track efficiency over time |
| 🧠 Re-Prompting | If JSON is invalid, re-prompt LLM with system: “Return valid JSON only.” | Can recover from partial hallucination |