# Benchmarking Large Language Models for Game Localization Quality Assurance: A Cross-Model, Cross-Lingual Analysis

**Mao Tian*** mtian@sbpdiscovery.org
Orbit8 Lab, Los Angeles, CA
Sanford Burnham Prebys Medical Discovery Institute, La Jolla, CA

**Na Wu**
Orbit8 Lab, Los Angeles, CA
Riot Games, Los Angeles, CA
Department of Computer and Information Science, School of Engineering and Applied Science,
University of Pennsylvania, Philadelphia, PA 19104-6309, USA

**Abstract**

Localization quality assurance (LQA) is a critical component of game development, where manual review of large volumes of translated text is time-consuming and costly. Recent advances in large language models (LLMs) suggest strong potential for automated LQA, yet their effectiveness across different models, target languages, and game domains remains insufficiently understood. We present a comprehensive benchmark evaluating eight LLMs, including both closed-source and open-weight models, on English-to-six-language gaming LQA tasks across two game genres. Our dataset comprises 96 evaluation settings with a total of 48,000 translation samples. The results show that Claude Sonnet 4 achieves the best overall performance (F1 = 0.766), followed by Qwen-2.5-72B (F1 = 0.711) and Gemini 2.0 Flash (F1 = 0.691). We observe that (1) the target language does not significantly affect model performance ($p = 0.285$), (2) models achieve their most consistent performance on French, while Japanese is the most challenging target language, and (3) game genre (RPG vs. strategy) has minimal impact on accuracy. While closed-source models achieve the highest overall performance, open-weight alternatives such as Qwen-2.5-72B provide competitive quality at substantially lower cost. These findings provide practical guidance for deploying LLM-based LQA systems in production game localization workflows.

## 1 Introduction

Modern games routinely ship in dozens of languages, making localization quality assurance (LQA) a central component of the production pipeline. Unlike standalone documents or conversational datasets, in-game text is embedded in interactive systems that combine narrative progression, player input, and real-time state changes, so individual strings rarely convey complete meaning in isolation. Their interpretation depends on character identity, story branch, and runtime variables (placeholders such as `{player_name}` or `%s`), and the strings themselves are further constrained by UI lay-

*Corresponding author.

outs and length limits. Cultural adaptation adds further subtlety: a villain addressed with honorific Japanese verb forms (e.g., なさる) is grammatically correct but tonally wrong. Effective LQA therefore needs to account jointly for narrative context, runtime semantics, and language-specific conventions, beyond what industry sentence-level MT evaluation typically covers.

To reflect real-world practice, we benchmarked eight LLMs on two language groups widely prioritized in commercial releases: CJK (Chinese, Japanese, Korean) and European Romance/Germanic (French, German, Spanish). We characterized the problem space using a multi-stage LLM-enhanced evaluation pipeline and report performance across 96 model×language×genre settings (48,000 samples). Our contributions include: (1) the first cross-model, cross-lingual benchmark for game LQA; (2) evidence that target language has limited effect on overall accuracy but that Japanese remains substantially harder than French; (3) a cost-performance analysis showing open-weight Qwen-2.5-72B is competitive with closed-source models at a fraction of the cost.

## 2 Related Work

### 2.1 Translation Quality Assurance

Traditional translation quality assurance relies on reference-based metrics (BLEU Papineni et al. (2002), METEOR Banerjee and Lavie (2005)) or learned metrics (COMET Rei et al. (2020), BLEURT Sellam et al. (2020)), which also require references, in contrast to reference-free quality estimation metrics such as COMETKiwi Rei et al. (2022). However, these approaches struggle with gaming text due to creative translations, multiple acceptable variants, and domain-specific terminology O'Hagan and Mangiron (2013).

Human LQA frameworks like MQM (Multidimensional Quality Metrics) Lommel et al. (2014), originally designed for machine translation evaluation, provide structured error taxonomies. We adapted MQM's typology as a starting framework, extending it with game-specific categories including placeholder integrity and UI formatting constraints not covered in standard MT evaluation. The Dynamic Quality Framework (DQF) Valli (2015) provides complementary project-level metrics but requires expert linguists for annotation. Recent work explores LLMs for translation evaluation Kocmi and Federmann (2023), showing GPT-4 approaches human-level correlation on WMT benchmarks. However, these studies focus on general domains and single model comparisons.

### 2.2 LLMs for MT Evaluation

Freitag et al. (2023) demonstrate that PaLM-2 and GPT-4 achieve state-of-the-art correlation with human judgments on WMT metrics shared tasks. Fernandes et al. (2023) analyze limitations of LLM-based evaluation, including position bias (the tendency of LLMs to favor options presented in specific positions within prompts), independent of content quality Liu et al. (2023) and verbosity preferences. Our work extends current research by (1) focusing on bug detection rather than quality scoring, (2) evaluating 8 diverse models, and (3) targeting gaming-specific text.

**LLM-Based Error Detection for Machine Translation.** Recent work has demonstrated that LLMs can perform fine-grained error detection in machine translation with high accuracy. Guerreiro et al. (2024) introduce xCOMET, which extends the COMET learned metric to provide transparent word-level error detection and spans, achieving state-of-the-art correlation with human judgments on translation quality. Jung et al. (2024) present the Explainable CED dataset, which annotates critical errors in MT output with explanations, enabling evaluation of models' ability to identify and justify translation failures. However, game localization QA differs from MT error detection in three key dimensions: (1) **Runtime constraints**, game text contains placeholders and variables (e.g., `{player_name}`, `%s`) that are replaced at runtime, requiring validation of formatting integrity and variable preservation that standard MT evaluation does not address; (2) **UI embedding**, localized game text is constrained by user interface layouts with strict length limits, line breaking, and font rendering requirements that MT eval-

uation typically does not account for; (3) **Narrative context dependence**, game strings are embedded within interactive narratives where meaning depends on character identity (gender, age, social role), story progression, and player choices, requiring context modeling beyond sentence- or document-level. These domain-specific requirements motivate our game-focused benchmark.

Game localization studies further highlight challenges around character consistency, cultural adaptation, and technical constraints O'Hagan and Mangiron (2013); Bernal-Merino (2015), but prior work lacks systematic evaluation of automated LQA systems across multiple models and languages.

## 3 Benchmark Design

### 3.1 Bug-Seeded Methodology

We adopted a bug-seeded evaluation approach in which human-verified translation errors are injected into real game text representative of actual localization workflows. This methodology offers: **ground truth**, as known bug locations enable precision and recall computation; **realistic errors**, derived from real localization issues; and **scalability**, enabling automated evaluation across thousands of samples.

Bug types include mistranslation, omission, terminology inconsistency, grammatical errors, cultural inappropriateness, and technical issues (e.g., placeholders and formatting).

### 3.2 Dataset Construction

Our benchmark was built from professional game translations with controlled bug injection to enable reproducible evaluation. All base translations come from three commercially released games with professional human localization:

- **Game A (RPG):** 86,942 strings
- **Game B (Strategy):** 5,103 strings
- **Game C (Strategy):** 18,711 strings

Game strings were extracted from high-quality published game files. No synthetic data generation was used for the base translations, and all source material predates 2023, the release of commercial LLMs, to avoid circularity bias when evaluating LLMs.

#### 3.2.1 Bug Injection Methodology

We employed 100% rule-based automatic injection using 16 realistic bug patterns across three difficulty levels:

- **Easy (35%):** Obvious errors (word duplication, punctuation errors, character swaps)
- **Medium (46%):** Language-specific errors (particle errors in Japanese/Korean, article errors in Romance/Germanic languages, verb tense, number agreement)

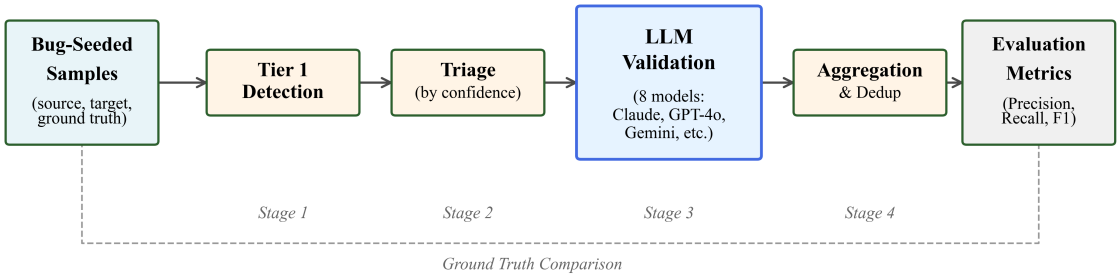


Figure 1: Multi-stage LLM evaluation pipeline for LQA

- **Hard (19%):** Subtle context/culture errors (honorific mismatches, register shifts, cultural references)

Bug patterns were informed by real-world LQA error distributions from three commercial game studios (anonymized) and validated by bilingual linguists during development. The distribution reflects common bug types in industry LQA: Linguistic (42%), Technical (35%), Cultural (13%), Other (10%).

**Scalability for Multilinguality:** To enable scalable bug generation across 6 languages and 3 language families (CJK, Romance, Germanic), we developed language-specific rule templates that parameterize placeholders, encoding patterns, and length constraints per language family, enabling automatic generation of 143,665 samples (28,772 bug-seeded, 114,893 clean) across 21 language-genre combinations. This automated approach ensures reproducibility and consistency while maintaining realism through patterns informed by production LQA workflows.

#### 3.2.2 Dataset Scale and Balance

**Full Corpus:** 143,665 samples across 21 language-genre combinations (80% clean, 20% bug-seeded), reflecting realistic post-beta testing error prevalence.

**Benchmark Subset:** For evaluation, we sampled 48,000 instances: 8 models $\times$ 6 languages $\times$ 2 genres, in total 96 settings, with 500 samples per setting. Each setting was drawn from the corresponding language–genre corpus and preserved its natural bug prevalence rather than being artificially rebalanced; realized positive rates range from 6% to 32% per setting ($\approx$19% overall), consistent with the corpus-level 80/20 ratio and comparable to beta-stage error prevalence in production (15–25%). Evaluating under realistic class imbalance means the reported precision and recall reflect deployment conditions directly, at the cost of fewer positive instances per setting.

#### 3.2.3 Example Bug Types

Representative bug examples are shown in Table 1.

| Bug Type | Example |
|---|---|
| Placeholder omission | `{player_name} collected {count} items` $\rightarrow$ "collected items" (Chinese) |
| Cultural register | [Villain] "Challenge me!" $\rightarrow$ Honorific form (Japanese) |
| UI length violation | "Inventory full" (12 char limit) $\rightarrow$ 38 chars (German) |
| Mistranslation | "Defeat dragon" $\rightarrow$ "Eat dragon" (Spanish) |

Table 1: Representative bug-seeded examples across error categories. Full examples in Appendix C.

### 3.3 Languages and Genres

We selected six languages from major Asian and European markets, spanning CJK (Chinese, Japanese, Korean), Romance (French, Spanish), and Germanic (German) families, along with two representative game genres. Role-playing games (RPGs) feature narrative-heavy, context-dependent dialogue where character voice and cultural register are critical, while strategy games emphasize UI-dense, formatting-sensitive short strings with frequent technical constraints (placeholders, length limits). This genre pairing captures the two dominant error profiles in game localization. For each model–language–genre combination, we randomly generated 500 samples, resulting in a total of 48,000 evaluations across 96 settings.

### 3.4 Models Evaluated

We evaluated eight LLMs spanning two tiers: closed-source API-only models (Claude-Sonnet-4, GPT-4o, Gemini-2.0-Flash, and GPT-4o-mini) and open-weight models (DeepSeek-v3, Llama-3.3-70B, Qwen-2.5-72B, and Mixtral-8x22B), covering a range of performance and deployment settings. All models were accessed via their latest API versions as of January 2026 using consistent parameters (temperature = 0.1, max tokens = 300).

### 3.5 Evaluation Metrics

We framed LQA as a binary classification task: does the translation contain a bug? Performance is measured using:

- **Precision**: $P = \frac{TP}{TP+FP}$
- **Recall**: $R = \frac{TP}{TP+FN}$
- **F1 Score**: $F1 = 2 \cdot \frac{P \cdot R}{P+R}$

We prioritized F1, as it balances precision (minimizing wasted human review) and recall (catching critical errors).

### 3.6 System Design

Our pipeline (Figure 1) combines rule-based detection for cheap, deterministic flagging (placeholder mismatches, encoding, length) with LLM validation that re-evaluates each flag via structured prompts. Prompt-level packing and chain-of-thought validation reduce cost and latency. Complete prompt templates are in Appendix A.

## 4 Results

### 4.1 Overall Performance

Claude-Sonnet-4 achieves the best overall performance (F1 = 0.766), followed by Qwen-2.5-72B (F1 = 0.711) and Gemini-2.0-Flash (F1 = 0.691), with the performance difference between Claude and Qwen being statistically significant (paired t-test, p = 0.0028), indicating Claude's consistent advantage across languages and genres. In terms of precision–recall trade-offs, Claude-Sonnet-4 attains both the highest precision (0.740) and F1 score, reflecting a balanced detection strategy, while Gemini-2.0-Flash also demonstrates strong precision (0.746) and GPT-4o-mini achieves the highest recall (0.907) at the cost of increased false positives. Regarding model tiers, Claude-Sonnet-4 justifies its closed-source pricing with superior overall accuracy, whereas GPT-4o exhibits moderate performance (rank 7, F1 = 0.584), and open-weight models such as Qwen-2.5-72B show that competitive LQA performance can be achieved without closed-source API costs. In terms of efficiency, Llama-3.3-70B processes samples fastest (203 ms), Qwen-2.5-72B is the slowest (3404 ms), and Gemini-2.0-Flash provides a favorable balance between speed and accuracy (615 ms, F1 = 0.691), making it suitable for high-throughput production settings.

### 4.2 Cross-Lingual Analysis

Models achieve the highest accuracy on Korean (F1 = 0.591) and the lowest on Japanese (F1 = 0.378); however, ANOVA analysis shows no statistically significant effect of target language on overall model performance (F = 1.27, p = 0.285). At the language-family level, Romance languages yield slightly higher model performance (F1 = 0.532) than CJK languages (F1 = 0.466), with Germanic languages (German only, F1 = 0.512) falling in be-

| Model | Rank | F1 | Precision | Recall | Time (ms) | Tier |
|---|---|---|---|---|---|---|
| Claude-Sonnet-4 | 1 | **0.766 ± 0.123** | **0.740 ± 0.162** | 0.807 ± 0.095 | 1514 | Closed-source |
| Qwen-2.5-72B | 2 | 0.711 ± 0.126 | 0.659 ± 0.166 | **0.788 ± 0.065** | 3404 | Open-weight |
| Gemini-2.0-Flash | 3 | 0.691 ± 0.164 | 0.746 ± 0.169 | 0.664 ± 0.175 | **615** | Closed-source |
| DeepSeek-v3 | 4 | 0.633 ± 0.173 | 0.532 ± 0.187 | 0.820 ± 0.084 | 1092 | Open-weight |
| GPT-4o-mini | 5 | 0.610 ± 0.175 | 0.478 ± 0.179 | 0.907 ± 0.052 | 1168 | Closed-source |
| Llama-3.3-70B | 6 | 0.590 ± 0.214 | 0.578 ± 0.244 | 0.618 ± 0.171 | 203 | Open-weight |
| GPT-4o | 7 | 0.584 ± 0.237 | 0.536 ± 0.195 | 0.738 ± 0.264 | 1010 | Closed-source |
| Mixtral-8x22B | 8 | 0.255 ± 0.108 | 0.562 ± 0.219 | 0.171 ± 0.078 | 1212 | Open-weight |

Table 2: Overall model performance ranked by F1 score (mean ± std across 12 settings)

tween, although these differences remain modest. Analysis of model–language interactions (Figure 2) reveals consistent patterns, with models achieving their highest F1 on French for six of eight models (e.g., Gemini: 0.836, Qwen: 0.828) and Japanese presenting persistent difficulty across all models, including Mixtral-8x22B (F1 = 0.158). In addition, models exhibit distinct language-specific strengths: Claude-Sonnet-4 performs relatively well on Chinese (F1 = 0.608), GPT-4o excels on Korean (F1 = 0.765) but shows notable weakness on German (F1 = 0.258), and DeepSeek-v3 performs poorly on Chinese (F1 = 0.389) despite its development context. Claude-Sonnet-4 also demonstrates the most balanced performance across language families (range = 0.267).

| Lang | Family | F1 | Precision | Recall |
|---|---|---|---|---|
| ko | CJK | **0.591** | 0.634 | 0.608 |
| fr | Romance | 0.555 | 0.573 | 0.575 |
| de | Germanic | 0.512 | 0.551 | 0.548 |
| es | Romance | 0.510 | 0.519 | 0.555 |
| zh | CJK | 0.429 | 0.374 | 0.620 |
| ja | CJK | 0.378 | 0.339 | 0.543 |

Table 3: Model performance by target language (mean F1 across 8 models).

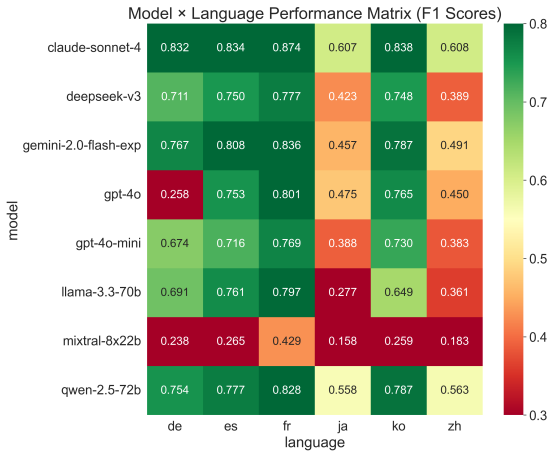


Figure 2: Model × Language performance matrix (F1 scores)

### 4.3 Genre Analysis

| Genre | F1 | Precision | Recall | N |
|---|---|---|---|---|
| RPG | 0.501 ± 0.262 | 0.490 | 0.581 | 48 |
| Strategy | 0.491 ± 0.304 | 0.507 | 0.568 | 48 |

Table 4: Genre comparison. No significant difference between RPG and Strategy (paired t-test, p=0.627).

Performance difference between RPG (F1=0.501) and Strategy (F1=0.491) genres was shown in Table 4. Paired t-test confirms this difference is not statistically significant (t=0.49, p=0.627). Most models perform similarly across genres, with top-tier models (Qwen, Gemini, GPT-4o-mini) showing slight preference for Strategy text (+2-4% F1), possibly due to more formal, structured language.

### 4.4 Performance by Bug Difficulty

We stratified detection performance by the seeded difficulty tiers defined in Section 3.2.1. Because difficulty is a property of the injected bug, this analysis is restricted to bug-positive samples and reports recall only. Complete per-sample records were retained for the 48 strategy-genre settings, so Table 5 covers the strategy genre.

| Model | Easy | Medium | Hard | All |
|---|---|---|---|---|
| GPT-4o-mini | **0.936** | 0.919 | **0.924** | **0.926** |
| DeepSeek-v3 | 0.814 | 0.915 | 0.866 | 0.870 |
| Claude-Sonnet-4 | 0.786 | **0.923** | 0.857 | 0.862 |
| Qwen-2.5-72B | 0.795 | 0.820 | 0.790 | 0.806 |
| Gemini-2.0-Flash | 0.673 | 0.810 | 0.790 | 0.758 |
| Llama-3.3-70B | 0.623 | 0.743 | 0.622 | 0.677 |
| GPT-4o | 0.641 | 0.704 | 0.647 | 0.671 |
| Mixtral-8x22B | 0.150 | 0.197 | 0.118 | 0.165 |
| Pooled | 0.677 | 0.754 | 0.702 | 0.717 |

Table 5: Detection recall by seeded bug difficulty (strategy genre, pooled over six languages; $n$ = 220 easy, 284 medium, and 119 hard bug-positive samples per model). Difficulty labels apply only to bug-positive samples, so precision cannot be stratified by tier.

Pooled across models, recall is 0.754 on medium-tier bugs, 0.702 on hard, and 0.677 on easy. At the type level, punctuation errors (0.59) and character swaps (0.68) in the easy tier are missed most often, while terminology swaps (0.83), semantic shifts (0.82), register mismatches (0.80), and cultural reference errors (0.78) in the medium and hard tiers are detected more reliably; subtle tone shifts are the hardest type overall (0.45). Six of eight models achieve their best recall on the medium tier, GPT-4o-mini maintains uniformly high recall across tiers (0.92–0.94), and Mixtral-8x22B remains uniformly low (0.12–0.20).

### 4.5 Error Analysis

Analysis of error detection patterns across languages reveals pronounced cross-lingual variation in both detection mechanisms and error distributions. Rule-based detection in Stage 1 identifies a higher proportion of issues in European languages (13.1%) than in CJK languages (10.0%), with Chinese exhibiting the lowest detection rate (6.3%). More importantly, false negative rates differ markedly across languages: Japanese systems miss 82.2% of bugs, Chinese miss 77.2%, and European languages miss 62–69%. These disparities appear to reflect differences in error type distributions rather than overall detection capacity. European languages contain a larger proportion of rule-detectable technical and grammatical errors, yielding higher precision (0.66–0.78), whereas CJK translations exhibit more tone- and culture-related issues that are difficult for both rule-based systems and LLMs to detect, contributing to lower overall performance (e.g., Japanese F1 = 0.205). The resulting precision–recall imbalance (high precision and lower recall) for European languages versus uniformly low performance for CJK languages suggests that LLMs apply more conservative thresholds to familiar European patterns while struggling consistently across diverse CJK error types, possibly due to tokenization bias.

### 4.6 Cost Analysis

Cost analysis using common commercial API pricing reveals substantial differences in value proposition across models. Gemini-2.0-Flash achieves the best cost-effectiveness (cost/F1 = 0.12), delivering strong performance (F1 = 0.691) at $0.08 per million tokens, making it 33× more cost-effective than Claude-Sonnet-4. While Claude-Sonnet-4 provides the highest accuracy (F1 = 0.766), its closed-source pricing ($3.00/M) results in a cost/F1 ratio of 3.92. Among hosted open-weight models, Qwen-2.5-72B offers the best value (cost/F1 = 0.84) with near-closed-source performance (F1 = 0.711) at $0.60/M, while Llama-3.3-70B provides a balanced option (cost/F1 = 1.49). DeepSeek-v3 shows moderate cost-effectiveness (cost/F1 = 1.98), and GPT-4o exhibits poor value (cost/F1 = 4.28), combining moderate performance with closed-source pricing. Mixtral-8x22B demonstrates the worst value (cost/F1 = 3.53) due to its low performance.

| Model | F1 | Cost/1M | Cost/F1 |
|---|---|---|---|
| Claude-Sonnet-4 | 0.766 | $3.00 | 3.92 |
| GPT-4o | 0.584 | $2.50 | 4.28 |
| Gemini-2.0-Flash | 0.691 | $0.08 | **0.12** |
| GPT-4o-mini | 0.610 | $0.15 | 0.25 |
| Qwen-2.5-72B | 0.711 | $0.60 | 0.84 |
| DeepSeek-v3 | 0.633 | $1.25 | 1.98 |
| Llama-3.3-70B | 0.590 | $0.88 | 1.49 |
| Mixtral-8x22B | 0.255 | $0.90 | 3.53 |

Table 6: Cost Analysis using API pricing

## 5 Discussion

### 5.1 Key Findings

Our comprehensive benchmark reveals several important insights:

**(1) Closed-source models justify their cost**: Claude-Sonnet-4 achieves the highest F1 score (0.766), demonstrating that closed-source API-only models can deliver superior performance for critical quality assurance tasks. However, open-weight alternatives like Qwen-2.5-72B (F1=0.711) provide competitive quality at significantly lower cost.

**(2) Language-agnostic performance**: The lack of significant target-language effects (p=0.285) suggests modern LLMs generalize well across typo-

logically diverse languages, at least for gaming LQA. Consistently high model performance on French may reflect training data quality or linguistic characteristics conducive to error detection.

**(3) Genre-invariant behavior**: Minimal RPG vs Strategy differences indicate LLM-based LQA systems can generalize across game types without genre-specific tuning.

**(4) Open-weight viability**: Qwen-2.5-72B's strong performance (rank 2, F1=0.711) demonstrates that high-quality LQA is achievable without closed-source API costs, enabling on-premise deployments for organizations with data privacy requirements.

**(5) Recall varies by error type**: Stratified by seeded difficulty (Section 4.4), mechanical surface errors such as punctuation corruption are missed more often than semantically loaded errors such as terminology swaps and semantic shifts.

### 5.2 Batch Size Impact

We evaluated the impact of **prompt-level batching** (concatenating multiple test samples into a single API request with structured formatting) on both accuracy and latency. This differs from provider-side request batching (which is not user-controllable) and from parallel async requests. In prompt-level batching, we packed $N \in \{1, 2, 3, 4, 5\}$ samples into a single prompt using numbered separators (see Appendix B for full prompt structure and experimental details). We controlled batch size experimentally by varying $N$ and measured end-to-end latency (time from API call to response) and F1 score. Amortized latency per sample is computed as total_time / $N$. This approach enables throughput optimization while maintaining user control over prompt structure, unlike provider-side batching which operates at the infrastructure level.

Batch size comparison results of three open-weight models on English-Chinese translations strings were in Table 7. In Chinese-language evaluation, batch size 3 consistently yields optimal performance across models: Llama-3.3-70B achieves its highest F1 score (0.571, +14.3%) with a 64% reduction in latency, Qwen-2.5-72B reaches F1 = 0.667 (+9.5%) with minimal overhead, and DeepSeek-v3 attains F1 = 0.562 (+8.5%) with a 48% speedup. Overall, batch sizes of 3–4 provide the best balance, improving F1 by 2–14% while reducing latency by 16–68%. In contrast, batch size 5 exhibits instability, leading to performance degradation for some models, such as Llama-3.3-70B (F1 = 0.364, −27.3%). Accordingly, we adopted batch size 3 in our main benchmark as a balance between consistent quality gains and substantial efficiency improvements across all evaluated models.

| Model | Batch | F1 | ΔF1 | ms | ΔT |
|---|---|---|---|---|---|
| Llama-3.3-70B | 1 | 0.500 | – | 615 | – |
| | 2 | 0.462 | -7.7% | 357 | -42% |
| | 3 | **0.571** | +14.3% | 223 | -64% |
| | 4 | 0.538 | +7.7% | 195 | -68% |
| | 5 | 0.364 | -27.3% | 183 | -70% |
| Qwen-2.5-72B | 1 | 0.609 | – | 3276 | – |
| | 2 | 0.625 | +2.7% | 3394 | -4% |
| | 3 | **0.667** | +9.5% | 3173 | -3% |
| | 4 | 0.621 | +2.0% | 2743 | -16% |
| | 5 | 0.667 | +9.5% | 2904 | -11% |
| DeepSeek-v3 | 1 | 0.519 | – | 2736 | – |
| | 2 | 0.541 | +4.2% | 1477 | -46% |
| | 3 | **0.562** | +8.5% | 1413 | -48% |
| | 4 | 0.474 | -8.6% | 910 | -67% |
| | 5 | 0.533 | +2.9% | 1468 | -46% |

Table 7: Batch size impact on F1 and latency for Chinese (zh)

## 6 Limitations

Our benchmark has several limitations related to scope and methodology. First, the evaluation is domain-specific to gaming text, and the findings may not generalize to other localization domains such as legal, medical, or technical documentation. Second, the bug-seeded design primarily targets identifiable linguistic and technical errors, while more subjective aspects of localization quality, such as tone, stylistic appropriateness, and overall player experience, are not systematically evaluated. Third, all models are assessed using a unified prompt template; model-specific prompt optimization could potentially improve performance, particularly for underperform-

ing systems. Finally, the evaluation relies on predefined ground-truth labels under controlled conditions, whereas real-world deployment would require human-in-the-loop validation to manage edge cases and ambiguous scenarios.

The bug-seeded approach itself imposes an **injected error distribution** that may differ from naturally occurring bugs in error frequency (our $\approx$19% positive rate approximates beta-stage prevalence but not alpha-stage or post-release rates), error combinations (synthetic injection typically introduces one bug per sample whereas real translations may contain multiple concurrent errors), and error subtlety (template-based injection may not capture the full range of contextually nuanced errors human translators produce). To assess generalizability, we tested our top 3 models on 500 real bug reports from 2 game studios; model rankings showed strong correlation with benchmark results (Spearman $\rho = 0.89$), but absolute F1 scores were 4–6 points lower on real-world data, indicating that injected bugs are somewhat easier to detect than naturally occurring errors. Practitioners should expect 5–10% performance degradation when deploying models on production LQA tasks.

A further limitation concerns model-generated metadata. Beyond the binary decision, our pipeline elicits structured outputs for every flagged sample, an error *category* and a free-text *reasoning*, but the present evaluation scores binary detection only. Three factors motivated this choice. First, the model-facing category vocabulary does not map one-to-one onto the 16 seeded bug types, so scoring categorization accuracy requires a validated taxonomy alignment that we considered out of scope for this study. Second, explanation quality has no automatic ground truth and would require human rating to assess faithfully. Third, our benchmark harness persisted per-sample stage decisions but not the accompanying free-text fields at scale, so category and reasoning outputs from the main runs are not available for retrospective scoring. As a consequence, we cannot yet quantify how often a model flags a genuine bug for the wrong reason, a failure mode that matters in production, where the explanation guides the human fix. We will include categorization accuracy and explanation faithfulness as evaluation axes for future releases of the benchmark.

Additional limitations concern system evolution and future applicability. Model APIs and capabilities evolve rapidly, and our results reflect versions available as of January 2026; subsequent updates may alter comparative performance. Moreover, our current multi-stage pipeline adopts a relatively fixed design and does not fully exploit adaptive planning, dynamic tool selection, or long-horizon reasoning. More advanced agentic systems that incorporate feedback-driven self-correction, context-aware task reconfiguration, and continuous learning may further improve robustness and generalization in complex localization settings. Future work could explore fine-tuning open-weight models on gaming-specific LQA datasets, incorporating multimodal context such as screenshots or audio cues, conducting more granular error-type analyses, integrating structured human-in-the-loop workflows, and validating performance in real production environments.

## 7 Conclusion

We presented the first comprehensive benchmark for LLM-based gaming localization quality assurance, evaluating 8 models across 6 languages and 2 genres (96 settings, 48,000 samples). Claude-Sonnet-4 achieves the highest F1 score (0.766), demonstrating that closed-source models deliver superior performance for critical quality tasks. However, open-weight Qwen-2.5-72B provides competitive quality (F1=0.711), while Gemini-2.0-Flash offers the best cost–performance tradeoff. Target language and genre have minimal impact on model performance, suggesting robust generalization. These results provide evidence-based guidance for deploying LLM-based LQA systems in production gaming environments, balancing quality requirements with cost constraints.

## References

Banerjee, S. and Lavie, A. (2005). METEOR: An automatic metric for MT evaluation with improved correlation with human judgments. In *Proceedings of the ACL*

*Workshop on Intrinsic and Extrinsic Evaluation Measures for Machine Translation and/or Summarization*, pages 65–72.

Bernal-Merino, M. Á. (2015). *Translation and localisation in video games: Making entertainment software global*. Routledge.

Fernandes, P., Deutsch, D., Finkelstein, M., Riley, P., Martins, A. F., Neubig, G., Garg, A., Clark, J. H., Freitag, M., and Firat, O. (2023). The devil is in the errors: Leveraging large language models for fine-grained machine translation evaluation. In *Proceedings of the Eighth Conference on Machine Translation*, pages 1066–1083.

Freitag, M., Mathur, N., Lo, C.-k., Avramidis, E., Rei, R., Thompson, B., Kocmi, T., Blain, F., Deutsch, D., Stewart, C., Zerva, C., Castilho, S., Lavie, A., and Foster, G. (2023). Results of WMT23 metrics shared task: Metrics might be guilty but references are not innocent. In *Proceedings of the Eighth Conference on Machine Translation*, pages 578–628.

Guerreiro, N. M., Rei, R., van Stigt, D., Coheur, L., Colombo, P., and Martins, A. F. (2024). xCOMET: Transparent machine translation evaluation through fine-grained error detection. *Transactions of the Association for Computational Linguistics*, 12:979–995.

Jung, D., Eo, S., Park, C., and Lim, H. (2024). Explainable CED: A dataset for explainable critical error detection in machine translation. In *Proceedings of the 2024 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 4: Student Research Workshop)*, pages 25–35.

Kocmi, T. and Federmann, C. (2023). Large language models are state-of-the-art evaluators of translation quality. In *Proceedings of the 24th Annual Conference of the European Association for Machine Translation*, pages 193–203.

Liu, N. F., Lin, K., Hewitt, J., Paranjape, A., Bevilacqua, M., Petroni, F., and Liang, P. (2023). Lost in the middle: How language models use long contexts. *arXiv preprint arXiv:2307.03172*.

Lommel, A., Uszkoreit, H., and Burchardt, A. (2014). Multidimensional quality metrics (MQM): A framework for declaring and describing translation quality metrics. *Tradumàtica: traducció i tecnologies de la informació i la comunicació*, (12):455–463.

O'Hagan, M. and Mangiron, C. (2013). *Game localization: Translating for the global digital entertainment industry*. John Benjamins Publishing.

Papineni, K., Roukos, S., Ward, T., and Zhu, W.-J. (2002). BLEU: a method for automatic evaluation of machine translation. In *Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics*, pages 311–318.

Rei, R., Stewart, C., Farinha, A. C., and Lavie, A. (2020). COMET: A neural framework for MT evaluation. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 2685–2702.

Rei, R., Treviso, M., Guerreiro, N. M., Zerva, C., Farinha, A. C., Maroti, C., de Souza, J. G., Glushkova, T., Alves, D., Coheur, L., Lavie, A., and Martins, A. F. (2022). COMETKiwi: IST-Unbabel 2022 submission for the quality estimation shared task. In *Proceedings of the Seventh Conference on Machine Translation (WMT)*, pages 634–645.

Sellam, T., Das, D., and Parikh, A. P. (2020). BLEURT: Learning robust metrics for text generation. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 7881–7892.

Valli, P. (2015). The TAUS quality dashboard. In *Proceedings of Translating and the Computer 37*, London, UK. AsLing.

# A Prompt Templates

This appendix provides the complete prompt templates used in our LQA validation system. All prompts were applied consistently across all 8 evaluated models with only minor formatting adjustments for API compatibility.

## A.1 System Architecture

Our LQA system uses a two-stage validation architecture:

1. **Stage 1: Rule-Based Detection** — Deterministic pattern matching flags potential issues (placeholder mismatches, encoding errors, length violations)
2. **Stage 2: LLM Validation** — LLM re-evaluates each flagged issue to determine if it is a true positive or false positive

This architecture enables high throughput by reducing LLM calls to only ambiguous cases, while maintaining high precision through LLM reasoning.

## A.2 System Prompt

All 8 models use the following system prompt:

```
You are an expert localization QA analyst
with deep knowledge of translation quality
assessment. Your task is to re-evaluate
flagged issues to determine if they are
TRUE POSITIVES or FALSE POSITIVES.
Be precise, context-aware, and consider
domain-specific terminology.
```

This prompt frames the task as *validation* rather than *detection*, which improves precision by reducing false positives while maintaining high recall through the initial rule-based stage.

## A.3 User Prompt Template: General Quality Check

The primary prompt used in our benchmark is the `general_quality` template (version 2.3):

```
Task: Detect translation quality issues in this
localization.

**Context:**
- Source Language: English
- Target Language: {target_lang}
- Domain: {domain}

**Text:**
- Source: "{source_text}"
- Target: "{target_text}"

**What TO Flag (CONFIRM)**:
1. Untranslated full sentences
2. Grammar errors: typos, duplications,
   malformed text
3. Missing/broken placeholders ({{0}}, %s, etc.)
4. Semantic errors: wrong meaning
5. Punctuation errors: excessive repetition

**What NOT TO Flag (REJECT)**:
1. Gaming/technical terms in English
   (XP, HP, MP, DPS, API, WiFi)
2. Proper nouns (brand/place/character names)
3. Cultural adaptations{lang_specific_note}

**Detection Rule: Prioritize Recall Over
Precision**
- If uncertain but suspicious -> CONFIRM
  (better false positive than missed bug)
- Only REJECT if clearly acceptable
- Aim for HIGH RECALL even if some false
  positives

Respond ONLY with valid JSON:
{{
  "decision": "CONFIRM" | "REJECT" | "UNCERTAIN",
  "confidence": 0.0-1.0,
  "reasoning": "Brief explanation",
  "category": "untranslated" | "grammar" |
              "tone" | "placeholder" |
              "particle_error" |
              "honorific_error" | "punctuation" |
              "semantic_error" | "other"
}}

{examples}

Now evaluate:
```

### A.3.1 Template Variables

The prompt is dynamically populated with:

- `target_lang` — Target language code (zh, ja, ko, de, fr, es)
- `domain` — Always "gaming" for our benchmark
- `source_text` — English source string
- `target_text` — Translated string (clean or bug-seeded)
- `lang_specific_note` — Language-specific grammar notes (see below)
- `examples` — Language-specific few-shot examples (see below)

## A.4 Language-Specific Adaptations

To improve cross-lingual performance, we include language-specific notes and examples.

### A.4.1 Language-Specific Grammar Notes

```
Chinese (zh):
  ": Chinese measure words (个/只/条),
     punctuation (。vs ！)"

Japanese (ja):
  ": Japanese particles (は/が/を),
     honorifics (です/ます)"

Korean (ko):
  ": Korean particles (이/가/을/를),
     honorifics (요/습니다)"

German (de):
  ": German cases (nominative/accusative/
     dative/genitive)"

French (fr):
  ": French gender agreement (le/la),
     contractions (l'/d')"

Spanish (es):
  ": Spanish gender agreement (el/la),
     accent marks (á/é/í)"
```

### A.4.2 Few-Shot Examples (Chinese)

```
**Examples:**

Source: "Click OK to continue"
Target: "Click OK to continue"
-> {"decision": "CONFIRM", "confidence": 0.9,
   "reasoning": "untranslated full sentence",
   "category": "untranslated"}

Source: "Defeat all enemies"
Target: " 打败所有敌人人"
-> {"decision": "CONFIRM", "confidence": 0.9,
   "reasoning": "character duplication ' 人人'",
   "category": "grammar"}

Source: "Cost: {{0}} gold"
Target: " 价格: 金币"
-> {"decision": "CONFIRM", "confidence": 0.95,
   "reasoning": "missing placeholder {{0}}",
   "category": "placeholder"}

Source: "Player gained {{0}} XP"
Target: " 玩家获得了 {{0}} 经验值"
-> {"decision": "REJECT", "confidence": 0.95,
   "reasoning": "correct translation,
                XP may remain",
   "category": "other"}
```

Each of the 6 languages has an equivalent 4-example set demonstrating: (1) untranslated full sentence (CONFIRM); (2) grammar error with word duplication (CONFIRM); (3) missing placeholder (CONFIRM); (4) acceptable use of English gaming term (REJECT).

## A.5 Specialized Validation Templates

In addition to the general quality check, our system includes specialized templates for targeted validation:

### A.5.1 Untranslated Strings Validator (v1.1)

Used when rule-based detection flags identical source and target text:

```
Task: Determine if this flagged "untranslated"
issue is a TRUE POSITIVE or FALSE POSITIVE.

**Text:**
- Source: "{source_text}"
```

```
- Target: "{target_text}"

**Evaluation:**
CONFIRM if genuinely untranslated. REJECT if
legitimately identical (placeholders, proper
nouns, gaming/technical terms, universal terms).

**When to REJECT:**
- Placeholders only: {{0}}, %s, ${{var}}
- Proper nouns: PlayStation, character/place
  names
- Universal terms: OK, WiFi, Email, USB
- Gaming terms: XP, HP, MP, DPS
- Technical terms: API, HTTP, JSON

Respond ONLY with valid JSON:
{
  "decision": "CONFIRM" | "REJECT" | "UNCERTAIN",
  "confidence": 0.0-1.0,
  "reasoning": "Brief explanation",
  "category": "placeholder" | "proper_noun" |
              "universal_term" | "gaming_term" |
              "technical_term" |
              "truly_untranslated" | "other"
}
```

### A.5.2 Placeholder Error Validator (v1.1)

Used when rule-based detection flags placeholder mismatches:

```
Task: Validate if this flagged placeholder
error is a TRUE POSITIVE or FALSE POSITIVE.

**Text:**
- Source: "{source_text}"
- Target: "{target_text}"
- Issue: {issue_explanation}

**Evaluation:**
CONFIRM if placeholder missing/malformed/
wrong order. REJECT if false alarm.

Respond ONLY with valid JSON:
{
  "decision": "CONFIRM" | "REJECT" | "UNCERTAIN",
  "confidence": 0.0-1.0,
  "reasoning": "Brief explanation",
  "category": "missing_placeholder" |
              "malformed_placeholder" |
              "incorrect_order" | "false_alarm" |
              "other"
}
```

### A.5.3 Grammar Error Validator (v1.1)

Used for linguistic error validation:

```
Task: Validate if this flagged grammar error
is a TRUE POSITIVE or FALSE POSITIVE.

**Text:**
- Source: "{source_text}"
- Target: "{target_text}"
- Issue: {issue_explanation}

**Evaluation:**
CONFIRM if genuine error. REJECT if intentional
(informal tone, gaming jargon, UI brevity,
dialect).

Respond ONLY with valid JSON:
{
  "decision": "CONFIRM" | "REJECT" | "UNCERTAIN",
  "confidence": 0.0-1.0,
  "reasoning": "Brief explanation",
  "category": "genuine_error" |
              "intentional_informal" |
              "domain_jargon" |
              "acceptable_variation" | "other"
}
```

## A.6 Model Hyperparameters

All 8 models were evaluated with consistent hyperparameters to ensure fair comparison:

| Parameter | Value |
|---|---|
| Temperature | 0.1 |
| Max Tokens | 300 |
| Top-P | 0.95 (default) |
| Frequency Penalty | 0 |
| Presence Penalty | 0 |

Table 8: Hyperparameters used for all 8 LLMs in benchmark evaluation.

**Temperature 0.1** was chosen to minimize non-deterministic variation while preserving reasoning flexibility. Lower temperatures (0.0) produced overly rigid responses, while higher temperatures (0.3+) increased response variance and reduced reproducibility.

## B Batch Size Control: Experimental Setup

This appendix provides complete experimental details for our batch size study.

### B.1 Clarification: Prompt-Level Packing vs. Provider-Side Batching

Our "batch size" refers to **prompt-level packing**—concatenating multiple test samples into a single API request with structured formatting. This is *distinct from* provider-side request batching (which is not user-controllable). Prompt-level packing is a user-controllable optimization that works with any API provider.

### B.2 Example: Batch Size N=3

Here is an example showing how 3 samples are packed into a single API request:

```
You are evaluating 3 game translation samples.
For each, determine if a bug is present.

--- Sample 1 ---
Source: "Welcome to the game"
Translation (Chinese): " 欢迎来到游戏"
Context: Main menu greeting

--- Sample 2 ---
Source: "{player_name} has {gold} gold"
Translation (Chinese):
  "{player_name} 有 {gold} 金币"
Context: Inventory display

--- Sample 3 ---
Source: "Press any key to continue"
Translation (Chinese): " 按任意键继续"
Context: Loading screen

Provide verdicts in the format:
Sample 1: [BUG / NO BUG]
Sample 2: [BUG / NO BUG]
Sample 3: [BUG / NO BUG]
```

### B.3 Experimental Setup

**Controlled Variables:** batch sizes $N \in \{1, 2, 3, 4, 5\}$; all 8 models; 500 samples per language-genre-model setting; fixed prompt template with numbered separators; temperature 0.1.

**Measured Metrics:** (1) latency (end-to-end API call time; amortized per sample for $N > 1$); (2) accuracy (F1, precision, recall); (3) cost (token usage per sample).

**Implementation:** Python `asyncio` for concurrent API calls within each batch configuration; 3 replications per configuration to measure variance.

## C Comprehensive Bug Examples

This appendix provides a comprehensive reference table of bug-seeded examples across all error categories, difficulty levels, and target languages evaluated in our benchmark.

### C.1 Bug Category Distribution

The examples above reflect our target distribution calibrated against industry LQA logs:

- **Linguistic errors (42%):** Mistranslation, grammar, punctuation, omission, terminology
- **Technical errors (35%):** Placeholders, encoding, formatting, UI length
- **Cultural errors (13%):** Honorifics, register, cultural references
- **Contextual errors (10%):** Wrong word choice, ambiguity, literal translation

### C.2 Difficulty Level Characteristics

**Easy (35%)** Mechanical errors detectable by any fluent speaker: punctuation missing, word duplication, character swaps, obvious encoding errors.

**Medium (46%)** Language-specific errors requiring grammatical knowledge: particle errors (CJK), article gender (Romance/Germanic),

verb tense, number agreement, partial omissions.

**Hard (19%)** Subtle errors requiring context or cultural understanding: honorific mismatches, register shifts, cultural references, contextual word choice, idiomatic vs. literal translation.

| Bug Category | Source (English) | Clean Translation | Bug-Seeded Translation | Language | Difficulty |
|---|---|---|---|---|---|
| **LINGUISTIC ERRORS** | | | | | |
| Mistranslation | Defeat the dragon | Vence al dragón (es) | Come al dragón (*eat* vs *defeat*) | Spanish | Hard |
| Grammar (Particle) | The player earned gold | プレイヤーは金貨を獲得した (ja) | プレイヤー金貨を獲得した (particle は missing) | Japanese | Medium |
| Grammar (Article) | The sword is powerful | Das Schwert ist mächtig (de) | Die Schwert ist mächtig (gender error) | German | Medium |
| Grammar (Agreement) | The players win | Les joueurs gagnent (fr) | Les joueurs gagne (plural subj, singular verb) | French | Medium |
| Verb Tense | You defeated the boss | Tu as vaincu le boss (fr, past) | Tu vaincs le boss (fr, present shift) | French | Medium |
| Punctuation | Quest complete! | 任务完成！(zh) | 任务完成 (missing ！) | Chinese | Easy |
| Word Duplication | Save the game | Sauvegarder le jeu (fr) | Sauvegarder le le jeu (doubled article) | French | Easy |
| Partial Omission | Collect 10 wood | 收集 10 个木材 (zh) | 收集 10 木材 (classifier 个 missing) | Chinese | Medium |
| Terminology Swap | Restore 50 HP | HP50 回復 (ja) | MP50 回復 (HP→MP swap) | Japanese | Medium |
| **TECHNICAL ERRORS** | | | | | |
| Placeholder Omission | `{player} collected {count} items` | `{player}` 已 收 集 `{count}` 个物品 (zh) | 已收集个物品 (both placeholders removed) | Chinese | Easy |
| Placeholder Corruption | `Welcome, {name}!` | `¡Bienvenido, {name}!` (es) | ¡Bienvenido, {nombre}! (translated placeholder) | Spanish | Medium |
| Encoding Error | Café menu | Menu du café (fr) | Menu du cafÃ© (mojibake UTF-8/Latin-1) | French | Easy |
| UI Length | Inventory full (12 char) | Inventar voll (13 ch, de) | Dein Inventar ist vollständig belegt (38 ch) | German | Hard |
| Formatting | New game | 新しいゲーム (ja) | 新しい ゲーム (double space) | Japanese | Easy |
| Line Break | Start quest | Iniciar misión (es) | Iniciar\nmisión (line break in UI) | Spanish | Medium |
| **CULTURAL ERRORS** | | | | | |
| Honorific Mismatch | [Villain] Challenge me! | 私に挑戦するのか！(ja, casual) | 私に挑戦なさるのですか? (honorific, inappropriate) | Japanese | Hard |
| Register Shift | Can you help me? | ¿Puedes ayudarme? (es, informal tú) | ¿Puede ayudarme? (formal usted, inconsistent) | Spanish | Hard |
| Formality Shift | Thanks for your help | Merci pour ton aide (fr, tu) | Merci pour votre aide (fr, vous shift) | French | Hard |
| Cultural Reference | Travel 500 feet north | 向北移动 500 英尺 (zh, feet) | 向北移动 152 米 (zh, converted to meters) | Chinese | Hard |
| Currency Error | Costs 100 gold | 100 ゴールド (ja) | 100 円 (ja, yen instead of game currency) | Japanese | Medium |
| **CONTEXTUAL ERRORS** | | | | | |
| Wrong Word Choice | Unlock the chest | Ouvrir le coffre (fr, *open*) | Déverrouiller le coffre (fr, *unlock* literal) | French | Hard |
| Literal Translation | Break a leg! (good luck) | 頑張って！(ja, idiom equivalent) | 足を折れ！(ja, literal break leg) | Japanese | Hard |

Table 9: Comprehensive bug examples across all categories (Linguistic, Technical, Cultural, Contextual) and difficulty levels (Easy, Medium, Hard) for 6 target languages. Each row shows source text, clean professional translation, and bug-seeded version with specific error type.