Midterm Final Project Check-in

YuChen (Jean) Lin lin.4842@osu.edu
Pakhi Chatterjee chatterjee.197@osu.edu
Aditya Pandey pandey.172@osu.edu

1 Problem Statement

Toxic comment detection models for Japanese are typically trained and evaluated on native-script text (kanji/kana). In practice, however, social text often appears in rōmaji (romanized Japanese) for ease of typing, stylistic effect, or to evade filters. Yet most open models are not evaluated for *script invariance*: whether predictions stay stable when only the script changes and the underlying Japanese sentence stays the same.

If a model flags 最低だ as toxic but misses the semantically identical "saiteida," moderation quality and fairness suffer. Recent Japanese toxicity resources exist, but they focus on native script; a paired, cross-script robustness evaluation is missing. We target that gap by building a lightweight script-invariant toxicity pipeline and a paired testbed that measures label stability when only the script changes.

2 Motivation

Japan has tightened penalties for online insults in 2022, making serious cases punishable by up to one year of imprisonment, so reliable toxicity detection in Japanese is no longer just an academic concern. At the same time, moderation quality is known to be uneven outside English, with evidence of under-investment and patchy language coverage across platforms. Romanized Japanese is ubiquitous because standard IMEs accept rōmaji keystrokes and convert them to kana/kanji; users can also intentionally leave text unconverted.

If safety systems only work reliably on native script, they will systematically miss toxic content written in rōmaji and treat some speakers less fairly than others. We therefore treat romanized Japanese as a *first-class input type* and ask whether modern models actually behave robustly under script changes.

3 Research Gap

Recent Japanese toxicity datasets such as LLM-jp's Japanese Toxicity Dataset (v2)[6] provide manually labeled toxic comments in native Japanese to support safer LLMs. Industry work on Japanese toxicity detection emphasizes efficiency and deployment concerns but similarly evaluates almost exclusively on native script. Parallel native/romanized benchmarks for Japanese toxicity are, to our knowledge, missing.

Romanized text has been recognized as a real failure mode in other languages. For example, Roman Urdu hate-speech detection work constructs dedicated datasets and models because romanization changes surface forms and challenges tokenizers [1]. A recent Nature-style line of work on multilingual toxicity [2] highlights script-agnostic robustness as a key requirement for safe moderation, but no equivalent study has focused on Japanese rōmaji.

Separately, modern NLP offers heterogeneous modeling approaches:

- **Language-specific subword models** such as Japanese BERT [3] are strong native-script baselines widely fine-tuned for Japanese tasks.

- **Multilingual subword models** such as mDeBERTa-v3 [4] are pre-trained on the CC100 corpus covering 100 languages, including Japanese, offering a cross-lingual baseline.
- **Tokenizer-free models** such as ByT5 [5] operate directly on raw UTF-8 bytes rather than on subword tokens, and are advertised as more robust to spelling noise and script issues.

However, we lack a systematic comparison of these model families on a script-controlled Japanese toxicity benchmark. This project fills that gap by building a paired native/rōmaji Japanese toxicity dataset and comparing how a language-specific subword model, a multilingual subword model, and a tokenizer-free byte-level model behave under script changes.

4 Research Questions and Hypotheses

We focus on two central research questions:

- **Script invariance**: For Japanese toxicity detection, how does model performance change when test inputs are converted from native script to rōmaji while keeping labels fixed?
- **Tokenization strategy**: Do tokenizer-free/byte-level models exhibit smaller performance drops and lower "flip rates" under romanization than subword models?

Our hypotheses are as follows:

- **H1 - Performance Gap:** Tokenized models (BERT Japanese, mDeBERTa) will show larger $|\Delta F_1|$ than ByT5 due to tokenization mismatch on romaji
- **H2 - Toxic Recall Preservation:** ByT5 will maintain higher toxic recall when switching to romaji compared to tokenized models
- **H3 - Perturbation Resilience:** ByT5 will show smaller F1 degradation under romanization variants than tokenized models

5 Technical Resources and Datasets

5.1 Models

We plan to fine-tune and compare three transformer models that differ primarily in their tokenization strategy:

- Japanese-specific subword model (Specialist)
  - **Base model**: $\mathrm{tohoku\text{-}nlp/bert\text{-}base\text{-}japanese\text{-}v3}$ [3]
  - A BERT variant pre-trained specifically on Japanese corpora, providing a strong native-script subword baseline.
- Multilingual subword model (Generalist)
  - **Base model**: $\mathrm{microsoft/mdeberta\text{-}v3\text{-}base}$ [4]
  - A multilingual DeBERTa-v3 model trained on the CC100 corpus, representing a high-capacity subword model with broad cross-lingual coverage.
- Tokenizer-free byte-level model
  - **Base model**: $\mathrm{google/byt5\text{-}small}$ [5]
  - A byte-level T5 variant that operates directly on UTF-8 bytes instead of subword tokens, designed to be robust to spelling and script variation.

Each model will be equipped with a standard classification head for strict binary toxicity prediction.

5.2 Training Datasets and Standardized Schema

We use two public Japanese toxicity datasets:

- llm-jp-toxicity-dataset-v2 [6]: native Japanese sentences with multi-annotator toxicity labels.
- inspection-ai/japanese-toxic-dataset [7]: native Japanese sentences with a three-way toxicity label and category flags.

Both are passed through a Python script which converts them into a single standardized CSV format with:

- one row per text (`id`, `text_native`),
- a unified four-way fine label (`Not Toxic`, `Hard to Say`, `Toxic`, `Very Toxic`),
- a three-way coarse label (`NonToxic`, `Ambiguous`, `Toxic`),
- and lightweight metadata (category list, original labels, basic confidence).

Currently we are mainly working with a Binary Strict view, where we drop rows whose coarse label is `Ambiguous` and keep only clearly non-toxic vs clearly toxic examples.

5.3 Paired Native/Rōmaji Views

From each standardized CSV, we create a paired native/rōmaji view using another Python script. For every example that survives the Binary Strict filter, we:

- keep the original `id`, labels, and `text_native`
- add a romanized version of the same text as `text_romaji`
- and save one processed file per source.

This produces two separate processed files, one for each dataset. Each row is therefore a matched pair (native vs rōmaji) with the same label, which is exactly what we need to evaluate script robustness.

5.4 Romanization Pipeline

We romanize Japanese using pykakasi (v2.2+) [8] with a simple, fixed policy:

- First normalize text with Unicode NFKC.
- Use Hepburn-style romanization.
- Do not insert word separators and do not use capitalization or macrons (long vowels are rendered in plain ASCII, e.g., “ou”, “oo”).

This gives us deterministic, ASCII-only rōmaji that roughly matches how Japanese is often typed online, and keeps the romanization behavior consistent across all experiments.

5.5 Compute Resources

- Single GPU instance (e.g., NVIDIA V100, 16 GB)
- Python 3.9, PyTorch, Hugging Face Transformers

6 Proposed Approach

6.1 Data Preprocessing and Validation

Our preprocessing has two separate scripted stages per dataset:

1. **Standardization**
   This is explained in detail under Section 5.2 and 5.3

2. **Binary + Pairing**
   We then run our pairing script on each standardized CSV. This script:

   - filters out rows with an `Ambiguous` coarse label (Binary Strict view),
   - generates `text_romaji` from `text_native` using the romanization pipeline above,
   - and writes the paired native/rōmaji CSVs in the processed data folder.

Some light sanity checks (spot-checking rows to make sure the rōmaji looks reasonable, and that IDs and labels are preserved) as a final data validation check.

6.2 Model Fine-tuning

- Train each model on both native Japanese and romanized (rōmaji) text independently to measure script-invariance
- Implement stratified train/test split (80/20) to maintain toxicity class distribution and ensure reproducibility via fixed random seed
- Use standard hyperparameters: learning rate 2e-5, batch size 16, 3 epochs, AdamW optimizer, dropout 0.1
- Save best model checkpoint based on validation accuracy for each configuration (model $\times$ script type)

7 Evaluation

Our evaluation strategy aims to validate our hypotheses and answer our research questions.

7.1 Script-Invariance Evaluation Framework

We evaluate all three models (mDeBERTa-v3, BERT Japanese, ByT5) using a unified set of metrics that quantify how well they maintain toxicity detection performance when switching from native Japanese to romaji:

- **Per-Script Performance:** Standard classification metrics (Precision, Recall, F1-score per-class and macro-averaged) computed separately for native and romaji test sets
- **Script-Invariance Gap:** $\Delta F_1 = F_{1,\text{native}} - F_{1,\text{romaji}}$ with 95% confidence intervals. Smaller $|\Delta F_1|$ indicates better script robustness.
- **Label Consistency Rate:** Percentage of test instances with identical predictions across native and romaji versions (Flip rate = 1 - Consistency)
- **Statistical Significance:** McNemar's test on paired native vs romaji predictions to assess whether error rate differences are statistically significant
- **Error Pattern Analysis:** Manual review of top 50 high-ΔF_1 cases to identify systematic failure modes

- **Efficiency Metrics (Optional):** Inference latency (ms/sample), throughput (samples/s), peak GPU memory

These metrics establish a baseline understanding of script-invariance performance for each model, which we then use in the comparative analysis below.

7.2 Tokenization vs Tokenization-free Model Comparison

Using the metrics from Section 7.1, we perform targeted comparative analysis to answer our second research question: does tokenization cause romaji performance degradation, and do byte-level models solve this problem?

7.2.1 Tokenization Failure Diagnostics

For tokenized models only (BERT Japanese, mDeBERTa), we measure tokenization-specific artifacts on romaji text:

- **Out-of-Vocabulary (OOV) Rate:** Percentage of romaji tokens absent from model vocabulary
- **Unknown Token Frequency:** Count of [UNK] tokens per sentence on romaji vs native
- **Tokenization Granularity:** Mean tokens/sentence ratio: $\frac{\text{tokens}_{\text{romaji}}}{\text{tokens}_{\text{native}}}$. Values > 1 indicate fragmentation.
- **Segmentation Errors:** For BERT Japanese, qualitative analysis of MeCab failures on romaji (e.g., incorrect morpheme boundaries)

7.2.2 Byte-Level Advantage Analysis

To verify whether ByT5's byte-level processing provides practical advantages:

- **Perturbation Resilience:** Measure $\Delta F_{1,\text{perturbed}} = F_{1,\text{clean}} - F_{1,\text{noisy}}$ on romaji with controlled noise (typos, spacing variations, vowel length ambiguity). Hypothesis: ByT5 shows minimal degradation.
- **Romanization Variant Robustness:** Compare error rates across romanization styles (Hepburn vs Kunrei-shiki approximations)

7.2.3 Cross-Architecture Analysis

- **Prediction Agreement:** Cohen's Kappa between tokenized models and ByT5 on romaji. $\kappa < 0.6$ suggests tokenization-induced disagreement; $\kappa > 0.8$ indicates agreement despite different architectures.
- **Error Taxonomy:** Categorize romaji test set predictions into four types:
  - Type A: All models correct (baseline difficulty)
  - Type B: Tokenized fail, ByT5 succeeds (tokenization failure evidence)
  - Type C: ByT5 fails, tokenized succeed (byte-level limitation)
  - Type D: All fail (inherently difficult)

  High Type B / Type C ratio supports tokenization as primary failure cause.
- **Computational Cost-Benefit:**
  - Measure inference latency (median, 95th percentile), GPU memory, throughput for all models
  - Calculate robustness gain per computational unit: $\frac{|\Delta F_{1\,\text{ByT5}}| - |\Delta F_{1\,\text{best-tokenized}}|}{\text{latency}_{\text{ByT5}} / \text{latency}_{\text{best-tokenized}}}$
  - Positive values indicate ByT5's robustness improvement justifies computational overhead

This two-stage evaluation design allows us to first establish individual model robustness, then systematically diagnose whether tokenization is the primary factor limiting romaji performance.

8 Actual Outcomes

This section summarizes what we have accomplished so far. We've completed the basic implementation and initial testing of our models. While we haven't finished everything we planned, we've made good progress and learned important things about how these models work with Japanese and romaji text.

8.1 Preliminary Results

We successfully implemented and evaluated three model architectures on paired native Japanese and romanized toxicity datasets using the llmjp dataset (3847 samples total, with 80/20 train/test split resulting in 769 test samples).

8.1.1 Script-Invariance Performance

Table 1 shows how each model performs on native Japanese text versus romanized (romaji) text. The key question is whether models can maintain the same toxicity detection accuracy when the only thing that changes is the script—not the meaning. We evaluate this using three metrics: overall accuracy, macro F1 (which treats toxic and non-toxic classes equally), and toxic recall (how well the model catches toxic content).

Model	Script	Accuracy	Macro F1	Toxic Recall	ΔF_1
mDeBERTa-v3	Native	0.91	0.90	0.87	-0.13
	Romaji	0.77	0.77	0.90	
BERT Japanese	Native	0.88	0.88	0.98	-0.12
	Romaji	0.78	0.76	0.61	
ByT5-small	Native	0.83*	-	-	TBD
	Romaji	0.17*	-	-	

*Quick test mode (50 samples, 1 epoch) - full evaluation pending

Table 1: Model performance on native Japanese vs romaji toxicity detection

8.1.2 Key Findings

- **Tokenized Models Show Script Degradation:** Both mDeBERTa and BERT Japanese experience 12-13 point drops in macro F1 when switching from native to romaji ($\Delta F_1 \approx -0.12$), providing preliminary evidence for H1 (tokenized models show larger performance gaps)
- **Different Failure Modes:**
  - *mDeBERTa on romaji*: Low non-toxic recall (0.69) but high toxic recall (0.90) - tends to over-predict toxicity
  - *BERT Japanese on romaji*: High non-toxic recall (0.89) but low toxic recall (0.61) - misses 39% of toxic content, likely due to MeCab tokenizer failures on romanized text
- **ByT5 Requires Further Investigation:** Preliminary results show poor performance on romaji (0.17 accuracy), suggesting training instability or implementation issues that need debugging before we can test H1-H3
- **Paired Datasets Established:** Successfully created paired native/romaji datasets from both inspection-ai (310 samples, for quick prototyping) and llmjp (3847 samples, used for reported results)

9 Future Work

We still have several important tasks to finish before we can fully answer our research questions. This section outlines what we need to do next, organized by the evaluation framework defined in Section 7.1 and Section 7.2.

9.1 Complete Script-Invariance Evaluation

- **Statistical Significance Testing:** Apply McNemar's test to determine if native vs romaji error rate differences are statistically significant for mDeBERTa and BERT Japanese
- **Label Consistency Analysis:** Calculate flip rates and consistency percentages across script changes
- **Perturbation Robustness:** Test all models on romanization variants (long-vowel representations, gemination styles, numeric leet)
- **Systematic Error Analysis:** Manual review of top 50 high-ΔF_1 cases to identify failure patterns

9.2 Complete Tokenization vs Tokenization-free Comparison

9.2.1 Immediate Priority: Fix ByT5 Training

- **Debug ByT5 Implementation:** Investigate the poor romaji performance (0.17 accuracy). Potential issues include:
  - Encoder-only usage in current implementation vs full encoder-decoder architecture
  - Learning rate or batch size unsuitable for byte-level models
  - Insufficient training epochs (currently 1 epoch in quick test, 3 in full mode)

  Implement proper T5 fine-tuning with decoder inputs or switch to ByT5-base encoder representations
- **Full-Scale ByT5 Training:** Train ByT5 on complete dataset with sufficient epochs to enable hypothesis testing (H1-H3)

9.2.2 Hypothesis Testing

We need to validate the hypothesis on ByT5 model because the current implementation only focuses on the tokenized models.

9.2.3 Tokenization Diagnostics

- **Vocabulary Analysis:** Measure OOV rates and [UNK] token frequency for BERT Japanese and mDeBERTa on romaji
- **Tokenization Granularity:** Calculate tokens/sentence ratios (romaji vs native) to quantify fragmentation
- **MeCab Segmentation Analysis:** Identify systematic MeCab failures on romanized text for BERT Japanese

9.2.4 Cross-Architecture Analysis

- **Prediction Agreement:** Calculate Cohen's Kappa between tokenized models and ByT5 on romaji predictions
- **Error Taxonomy:** Implement Type A/B/C/D categorization to quantify when tokenization vs byte-level processing fails
- **Computational Cost-Benefit:** Measure inference latency, GPU memory, throughput; calculate robustness gain per computational unit

10 Expected Final Outcomes

This section describes what we hope to achieve by answering our research questions and how our work will contribute to Japanese NLP researhc and practical toxicity detection systems.

10.1 Research Contributions

- **Script-Invariance Benchmark:** Comprehensive paired native/romaji evaluation for Japanese toxicity detection, demonstrating whether current models maintain fairness across script changes
- **Tokenization vs Tokenization-free Comparison:** Quantitative evidence on whether byte-level models (ByT5) outperform subword tokenized models (BERT Japanese, mDeBERTa) on romanized text
- **Failure Mode Taxonomy:** Systematic characterization of when and why tokenization breaks down on romaji (MeCab segmentation errors, OOV rates, [UNK] tokens) vs when byte-level processing fails

10.2 Practical Deliverables

- **Reproducible Training Pipeline:** Open-source code supporting multiple model architectures with script selection, enabling researchers to extend this work
- **Paired Datasets:** Two-tier benchmark system with native Japanese and Hepburn romaji versions:
  - Small-scale (inspection-ai): 310 samples for quick prototyping and testing
  - Large-scale (llmjp): 3847 samples for production-grade model training
- **Model Performance Baselines:** Published results for mDeBERTa-v3, BERT Japanese, and ByT5 on both native and romaji, establishing performance expectations for future work

References

[1] Hammad Rizwan, Muhammad Haroon Shakeel, and Asim Karim. 2020. Hate-Speech and Offensive Language Detection in Roman Urdu. *In Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 2512–2522, Online. Association for Computational Linguistics. https://doi.org/10.18653/v1/2020.emnlp-main.197

[2] Ashiq, W., Kanwal, S., Rafique, A. et al. Roman urdu hate speech detection using hybrid machine learning models and hyperparameter optimization. Sci Rep 14, 28590 (2024). https://doi.org/10.1038/s41598-024-79106-7

[3] Tohoku NLP. (n.d.). *bert-base-japanese-v3*. Hugging Face. https://huggingface.co/tohoku-nlp/bert-base-japanese-v3

[4] Microsoft. (n.d.). *mdeberta-v3-base*. Hugging Face. https://huggingface.co/microsoft/mdeberta-v3-base

[5] Google. (n.d.). *byt5-small*. Hugging Face. https://huggingface.co/google/byt5-small

[6] LLM-jp Consortium (2024). *Japanese Toxicity Dataset v2*. https://gitlab.llm-jp.nii.ac.jp/datasets/llm-jp-toxicity-dataset-v2

[7] Inspection AI (2024). *Japanese Toxic Dataset*. https://github.com/inspection-ai/japanese-toxic-dataset

[8] pykakasi Development Team (2024). *pykakasi: Japanese text transliteration library*. https://pypi.org/project/pykakasi

[9] Oikawa, Yuto, Yuki Nakayama, and Koji Murakami. 2022. *A Stacking-based Efficient Method for Toxic Language Detection on Live Streaming Chat.* In Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing: Industry Track, pages 571–578, Abu Dhabi, United Arab Emirates. Association for Computational Linguistics. https://doi.org/10.18653/v1/2022.emnlp-industry.58