Research Report: Evaluation of `test.py` and `test1.py`

Project: Himalayan GPT / Restaurant Ordering Research

Date: 22 Jul 2026

Workspace: /home/venom/Desktop/food_restaurant/himalayan-gpt

1. Executive Summary

Two scripts were evaluated. `test.py` performs a structured intent-classification experiment for a Nepali restaurant assistant using the model `himalaya-ai/himalaya-gemma-4-e2b-it`, while `test1.py` is a minimal free-generation smoke test using `himalaya-ai/himalayagpt-0.5b`. The first experiment demonstrates that the larger model can follow a strict JSON classification prompt with 92.31% accuracy on a small hand-crafted evaluation set, but inference latency is very high at about 39.60 seconds per request. The second experiment reveals a major generation-quality issue: the model fails to answer the question "nepal ko rajdhani?" and instead degenerates into repetitive text. Together, the results show promise for instruction-following classification, but they also highlight reliability, latency, and evaluation-depth gaps that must be addressed before production use.

2. Methodology

`test.py` loads a causal language model, applies a restaurant-intent system prompt, runs a warmup request, evaluates 13 labeled Roman Nepali test utterances, records latency and system/process/GPU resource snapshots, writes a JSON metrics file, and then opens an interactive loop for additional prompts.

`test1.py` loads a smaller causal language model and performs a single direct text generation for the prompt "nepal ko rajdhani?" without any evaluation harness, resource instrumentation, prompt constraints, or answer validation.

3. Runtime Environment

- Platform: Linux-7.0.0-1-cachyos-x86_64-with-glibc2.43
- Python: 3.11.14
- PyTorch: 2.13.0+cu130
- CUDA: 13.0
- Primary device: CUDA
- Numerical dtype in `test.py`: bfloat16
- GPU: NVIDIA GeForce RTX 4070 Laptop GPU
- GPU total memory: 7807.56 MB
- CPU logical cores: 16
- CPU physical cores: 8

Both terminal runs also reported unauthenticated Hugging Face Hub access. This does not invalidate the results, but it may reduce download performance and can create rate-limit risk for repeated experiments.

4. Findings from `test.py`

The script is designed as a controlled intent-classification experiment. It defines 11 intent labels, uses a strict system prompt, forces deterministic generation (`do_sample=False`), and parses the model output into JSON. This makes the experiment significantly more research-ready than the simple generation test in `test1.py`.

4.1 Accuracy and classification behavior

Summary metrics:

- Model: himalaya-ai/himalaya-gemma-4-e2b-it
- Accuracy: 0.9231 (12 correct out of 13)
- Mean latency: 39.6046 s
- Min latency: 32.7020 s
- Max latency: 46.6634 s

Utterance	Expected	Predicted	Confidence	Result
2 momo deu	place_order	place_order	0.95	OK
ek plate chowmein dinus	place_order	place_order	0.95	OK
arko coke add gara	add_item	add_item	0.95	OK
ani fries pani deu	add_item	add_item	0.95	OK
fries hataidinu	remove_item	remove_item	0.95	OK
pizza spicy kam gara	modify_item	modify_item	0.95	OK
momo ko price kati ho	ask_price	ask_price	0.95	OK
menu pathau	ask_menu	ask_menu	0.95	OK
same order feri gara	repeat_order	repeat_order	0.95	OK
order cancel gara	cancel_order	cancel_order	0.95	OK
thik cha confirm gara	confirm_order	confirm_order	0.95	OK
namaste	greeting	greeting	0.95	OK
today weather kasto cha	unknown	greeting	0.95	MISS

Interpretation: the model handled restaurant-related intents well on the provided examples, including the difficult boundary between `place_order` and `add_item`. However, it failed to reject an unrelated weather query and instead labeled it as `greeting`. This is an important robustness gap because real users often ask off-topic or mixed-context questions.

4.2 Resource utilization during model load

- Model load wall time: 11.8557 s
- CPU utilization snapshot: 3.7%
- System RAM used: 9736.58 MB
- System RAM available: 5542.77 MB
- System RAM percent: 63.7%
- Process RSS memory: 3406.51 MB
- Process VMS memory: 21392.77 MB
- GPU memory allocated: 2284.97 MB
- GPU memory reserved: 2340.00 MB
- GPU free memory: 5287.62 MB
- Torch threads: 8

4.3 Final resource snapshot after evaluation

- CPU utilization snapshot: 1.2%
- System RAM used: 6830.42 MB
- System RAM available: 8448.92 MB
- System RAM percent: 44.7%
- Process RSS memory: 416.25 MB
- Process VMS memory: 23165.98 MB
- GPU memory allocated: 2293.09 MB
- GPU memory reserved: 2350.00 MB
- GPU free memory: 5261.62 MB

Interpretation: the experiment fits comfortably within the available 8 GB-class laptop GPU memory budget, but it achieves this partly through offloading. The terminal explicitly states that some parameters were placed on the meta device and offloaded to disk and CPU. This likely contributes to the very high per-query latency.

4.4 Interactive run observations

Observed interactive outputs:

- Input: malai noodles dinu na -> intent=place_order, confidence=0.95, latency=36.9522 s
- Input: tesma piro ra amilo halka dherai haldinus la -> intent=place_order, confidence=0.95, latency=40.8796 s
- Input: order done gardinus yeti ho -> intent=confirm_order, confidence=0.95, latency=36.0318 s

The second interactive example is notable. The user text appears to modify an existing item by changing flavor characteristics (spicy/sour intensity), yet the model predicted `place_order` and produced a questionable rationale ("placing a new order for tea and amilo"). This suggests that although the scripted benchmark accuracy is high, the model still struggles with context-sensitive attribute modification in free-form interactive use.

5. Findings from `test1.py`

`test1.py` performs a single unconstrained generation using `himalaya-ai/himalayagpt-0.5b`. The script does not impose a task format, does not request a concise answer, and does not enforce any evaluation criteria.

Observed terminal output:

Prompt: "nepal ko rajdhani?"
Output: "nepal ko rajdhani? ko ko ko ko ko ko ko ..."

Interpretation: the model failed a very simple factual query and entered repetitive degeneration. This indicates poor answer grounding for the tested prompt configuration and/or insufficient decoding constraints for stable output quality.

- Model: himalaya-ai/himalayagpt-0.5b
- Weight loading progress observed: 113 shards/items completed
- Execution time visible in terminal: about 14 seconds total
- Deprecation warning observed: `torch_dtype` is deprecated; `dtype` should be used instead
- No explicit RAM, CPU, or GPU metrics were collected by the script

6. Comparative Analysis

The two scripts demonstrate different research maturity levels. `test.py` is instrumented, reproducible, and narrowly scoped to intent classification; consequently, it provides usable quantitative evidence. `test1.py` is a quick functional probe rather than a proper experiment, so its output is useful mainly as a qualitative failure case. The larger Gemma-based model shows reasonable task performance under strong prompting, while the smaller model in the second script shows clear instability even for a basic QA prompt.

7. Limitations of the Current Research

- The evaluation set in `test.py` contains only 13 examples, which is far too small to estimate real-world performance reliably.
- The utterances are hand-crafted and may not reflect the diversity, ambiguity, spelling noise, code-switching, or multi-turn context seen in actual restaurant chats.
- The confidence score is almost always 0.95, which suggests the model is not well calibrated for uncertainty estimation.
- The benchmark is largely single-turn and does not test whether the model can track conversation state across multiple order updates.
- Only one clear failure case (`unknown` vs `greeting`) appears in the benchmark, so error analysis is incomplete.
- Interactive outputs reveal issues not visible in the benchmark, indicating that the scripted test set may be too easy or too narrow.
- For `test1.py`, there is no measurement framework, no baseline comparison, no prompt variants, and no resource instrumentation.

- Neither script compares multiple models, decoding settings, quantization strategies, or prompt templates, so the results do not yet support a strong model-selection conclusion.

8. Risks and Deployment Concerns

- High latency: 33 to 47 seconds per request is too slow for a production chat ordering assistant and would likely frustrate users.
- Misclassification risk: an off-topic message being interpreted as a valid restaurant intent can lead to wrong downstream behavior.
- Order-state risk: the model may confuse new orders with item modifications or additions when conversational context is implicit.
- Reasoning transparency risk: the generated `reason` field can sound plausible even when the prediction is wrong, which may mislead operators during debugging.
- Model offloading risk: CPU/disk offloading reduces memory pressure but may cause unstable performance across different machines or workloads.
- Repetition and generation-collapse risk: the behavior in `test1.py` shows that unconstrained generation can produce low-quality or unusable responses.
- Operational risk: unauthenticated Hugging Face downloads may cause rate limits or slower repeated experiment setup.

9. Improvement Recommendations

- Expand the evaluation set from 13 examples to at least several hundred examples covering all intents, spelling variations, ambiguous cases, off-topic inputs, and multi-turn follow-ups.
- Create a confusion matrix and per-intent precision/recall/F1 rather than relying only on overall accuracy.
- Add explicit conversation-state inputs so the model can distinguish `place_order`, `add_item`, and `modify_item` more reliably in multi-turn sessions.
- Introduce stronger unknown-detection examples and adversarial prompts to reduce false positives on unrelated text.
- Measure peak GPU memory, average throughput, and percentile latencies across repeated runs to obtain more robust performance characterization.
- Test smaller/quantized variants or optimized inference backends to reduce latency substantially.
- For `test1.py`, replace raw generation with task-specific prompts and safer decoding settings, and add objective checks for factual correctness.
- Update deprecated `torch_dtype` usage in `test1.py` to `dtype` for forward compatibility with newer `transformers` versions.
- Consider constrained decoding or classification heads for the intent task instead of open-ended generation, which could improve both speed and reliability.
- Add logging for failures, malformed JSON, and interactive misclassifications so qualitative error analysis becomes easier over time.

10. Conclusion

The current experiments suggest that the Gemma-based setup in `test.py` is a workable research baseline for Nepali restaurant intent classification, especially because it already includes reproducible prompts, deterministic decoding, and system resource measurements. However, the observed latency is very high and the experiment remains too small to support strong claims about deployment readiness. Meanwhile, `test1.py` demonstrates that the smaller model and prompt configuration used there are not reliable for even a simple factual task. Future work should focus on larger and more realistic evaluation data, better uncertainty handling, explicit multi-turn context modeling, and much faster inference.

Appendix A. Key Raw Outputs

`test.py` benchmark miss:

[MISS] 'today weather kasto cha' -> greeting (expected=unknown, conf=0.95, 38.3682s)

`test1.py` raw generation:

nepal ko rajdhani? ko ko ko ko ko ko ko ko ko ko ko ko ko ko ko ko ko ko ko ko ko ko ko ko ko ko ko ko ko ko ko ko ko ko ko ko ko koko ko ko ko ko ko ko ko ko ko ko ko ko ko ko ko ko ko ko ko ko ko ko ko ko ko