LLMorph: Automated Metamorphic Testing of Large Language Models

Steven Cho
University of Auckland
Auckland, New Zealand
steven.cho@auckland.ac.nz

Stefano Ruberto
JRC European Commission
Ispra, Italy
stefano.ruberto@ec.europa.eu

Valerio Terragni
University of Auckland
Auckland, New Zealand
v.terragni@auckland.ac.nz

***Abstract*—Automated testing is essential for evaluating and improving the reliability of Large Language Models (LLMs), yet the lack of automated oracles for verifying output correctness remains a key challenge. We present LLMorph, an automated testing tool specifically designed for LLMs performing NLP tasks, which leverages Metamorphic Testing (MT) to uncover faulty behaviors without relying on human-labeled data. MT uses Metamorphic Relations (MRs) to generate follow-up inputs from source test input, enabling detection of inconsistencies in model outputs without the need of expensive labelled data. LLMorph is aimed at researchers and developers who want to evaluate the robustness of LLM-based NLP systems. In this paper, we detail the design, implementation, and practical usage of LLMorph, demonstrating how it can be easily extended to any LLM, NLP task, and set of MRs. In our evaluation, we applied 36 MRs across four NLP benchmarks, testing three state-of-the-art LLMs: GPT-4, LLAMA3, and HERMES 2. This produced over 561,000 test executions. The results demonstrate LLMorph's effectiveness in automatically exposing incorrect model behaviors at scale.**

The tool source code is available at https://github.com/steven-b-cho/llmorph. A screencast demo is available at https://youtu.be/sHmqdieCfw4.

***Index Terms*—large language models, metamorphic testing, machine learning testing, NLP, Software Engineering for AI**

I. Introduction

Large Language Models (LLMs) have seen widespread adoption across various domains thanks to their strong performance in natural language understanding and generation tasks [1]. Despite this success, serious concerns remain regarding their reliability and trustworthiness (e.g., bias, hallucination) [2]. Identifying these problems is critical to effectively addressing and mitigating them. Therefore, automated testing of LLM is crucial for evaluating the quality of LLM outputs. Moreover, automated testing of LLMs is becoming more important in industry, as many companies are moving away from using public LLM services and instead are running their own fine-tuned models locally [3]. This change is often due to concerns about privacy, security, and legal rules, or simply the need for better results in their specific tasks [3].

One of the primary challenges in automatically testing LLMs is the **oracle problem** [4]; *the problem of distinguishing between correct and incorrect test executions*. In **Natural Language Processing (NLP)**, generating new test inputs is relatively easy because large amounts of text data are readily available. However, checking whether the model's outputs are correct given an NLP task is much harder. This task usually relies on human-annotated labels, which act as "oracle" to judge correctness. Since manually creating these labeled datasets is time-consuming and costly, there is a strong need for automated oracles that can evaluate output quality without depending on human-generated labels.

Metamorphic Testing (MT) [5] is a widely used technique for addressing the oracle problem in software testing. Instead of requiring expected results for every test case, MT relies on **Metamorphic Relations (MRs)**: expected relationships among the outputs of related inputs. The key idea is that, even if one cannot automatically determine whether a single output is correct, *we can use the relationships among the expected outputs of multiple related inputs as a test oracle* [5]. An MR describes how the output should change (or remain consistent) when the input is modified in a specific way. For example, if two input texts are paraphrases of one another, then their outputs should also be similar. If this expected relationship is violated, the system may have produced an incorrect result.

To apply MT in practice, testing begins with a source input. This input is then transformed to create one or more follow-up inputs that satisfy the conditions of a given MR. The system under test is executed on both the original and the transformed inputs, and their outputs are compared. If the outputs do not satisfy the expected relationship defined by the MR, the test reports a failure. Thus, MT can detect faults even though the exact ground truth labels are not required for either case. A key advantage of MT is that the same MRs can be applied to many different inputs, enabling automated testing at large scale. This is particularly useful for testing LLMs, where failures often emerge only under specific input conditions [2], and where large volumes of unlabeled data are available.

Although MT has been applied across many NLP tasks, its use with LLMs remains relatively understudied [6]. To fill this gap, we recently presented the most comprehensive study on MT for LLMs in NLP to date. Our study was recently accepted at ICSME 2025 [7]. We conducted the first systematic literature search on MRs for NLP, reviewing 1,024 papers and identifying 44 that explicitly defined MRs. This resulted in a catalog of 191 unique MRs across 24 tasks. We also presented **LLMorph**, an automated test tool for MT on LLMs, implementing 36 of the 191 MRs across four tasks.

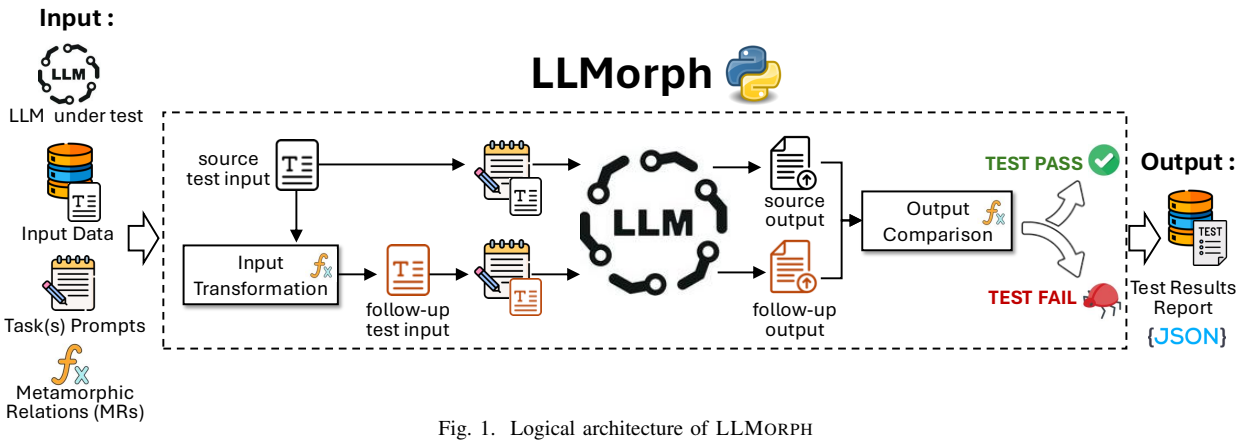

Fig. 1. Logical architecture of LLMorph

LLMorph was originally produced as a means to conduct our study [7]. In this demonstration paper, we focus specifically on the tool itself, giving details on its design, implementation and usage. The intended users of LLMorph are researchers and developers who wish to verify the robustness of an LLM system for the purpose of verification or improvement. Given an LLM and a list of test inputs, LLMorph produces a list of failing metamorphic test pairs, allowing users to identify potentially unknown faults. The source code of LLMorph can be found at: `https://github.com/steven-b-cho/llmorph`

II. LLMorph

The tool leverages metamorphic testing to measure the robustness of an LLM system. It enables users to apply metamorphic relations to any natural language task of their choice. Figure 1 illustrates the logical architecture of the tool.

A. Input

LLMorph takes as input four items:

LLM under test. Currently, LLMorph supports LLMs via any endpoint compatible with the OpenAI API format[1], and thus it is not necessary to host a local model. However, the tool can easily be extended to work with other APIs as well as locally deployed LLMs, if desired.

Input data. The input test data is structured as a list of textual source inputs of arbitrary length. Unlike in traditional testing, the inputs in LLMorph do not need to be labelled, as it is a metamorphic testing tool. This allows for a much larger amount of data to be used, since the cost associated with labelling the data is no longer a concern.

Task(s) prompts. The user can choose which NLP tasks to test the LLM on. LLMorph currently supports four built-in tasks: context-based question answering (QA), natural language inference (NLI), continuous sentiment analysis (SA), and relation extraction (RE). These tasks are implemented using zero-shot prompts to the LLM under test. Users can easily extend the tool by adding new tasks or modifying the existing ones through simple JSON files, which define the prompts used for each task. This is particularly useful when testing a fine-tuned LLM on any arbitrary task.

Metamorphic Relations (MRs). The user can choose which MRs to test the LLM using. Some relations can be used in multiple tasks, while others are specific to only one. In our recent work, we performed a systematic literature review and identified 191 MRs for NLP [7]; of these, LLMorph currently implements 36 (see [7] for more details). This subset was chosen based on commonality, applicability across different tasks, specificity to individual tasks, and ease of understanding.

B. Output

LLMorph produces as output a test result report, which is a JSON file containing information about all test runs. It includes the original source inputs, the generated follow-up inputs, the corresponding source and follow-up outputs produced by the LLM under test, and whether the metamorphic relation was violated or not in each case.

C. Process

For each combination of source input, task, and MR, LLMorph uses the input transformation specified by the MR to produce a follow-up input. Both inputs are processed through the same LLM under test using the task prompt to produce the respective source and follow-up outputs. These are then compared to determine whether they satisfy the output relation specified by the MR.

Example. Given GPT-4o as the **LLM under test**, and the **source test input**: `"The area in which a glacier forms is called a cirque. What geological features formed by glaciers?"`, let us assume the user selects the Question Answering (QA) **task** and the **metamorphic relation**: `"Adding random spaces to the input should not change the output."`

LLMorph begins by applying the transformation associated with the MR, resulting in the follow-up input: `"Th e a rea in wh ich a gl acier forms is cal l ed a ci rque. Wha t geologi cal f eatures form ed by glaci ers?"`

The QA task is defined by the following prompt:

[1]https://platform.openai.com/docs/api-reference/

```
Here is some information: "INPUT_0"
Using only this information,
nothing else, answer the following
question: "INPUT_1" Keep your
answer to a short sentence. If
you cannot give an answer, write
'unknown'.
```

Given this prompt, the LLM is queried twice: once with the original input and once with the transformed input. The **source output** is `unknown`, the **follow-up output** is `cirque`.

The **output comparison** based on the MR expects that both outputs should be the same. Since they differ, the MR is violated, and a faulty execution is detected. Note that we can find this real fault in GPT-4o without the need for labeled data specifying that `unknown` should be the correct output[2].

This simple example illustrates how LLMORPH automatically detects bugs in LLMs using MT. LLMORPH supports a broader set of more complex (and potentially more insightful) MRs across three additional, more advanced NLP tasks. A key strength of LLMORPH is its extensibility: additional metamorphic relations and tasks can be easily incorporated by simply adding to corresponding files.

D. Implementation Details

LLMORPH is implemented as a PYTHON3 project. It uses the `openai` library for communication with the LLM, the `sentence_transformers` library for semantic similarity calculation, and the `nlpaug` library to implement some MRs.

The implementation of the MRs are, on the whole, derived from semantic, non-precise definitions. The relations implemented currently have the following properties: An *input transformation* function that takes a source input and transforms it to create a follow-up output; a *output comparison* function that takes two outputs and compares them to determine whether they satisfy the MR's output relation; and (optionally) a set of *verifications*, which are restrictions to the inputs and outputs that must hold for the MR to be valid. This latter property is not how MRs are typically definitionally structured; however, it was done this way for ease of implementation. We implement the MRs through both function-based and LLM-based means. Simpler relations, such as *MR-84: Concatenating a random sentence*, are done through traditional functions, or a library. More complex MRs – for instance, *MR-51: Paraphrasing* – are implemented using a few-shot prompted LLM. The use of LLMs to test LLMs is a commonly used technique, enabling processes which would be much harder to achieve through traditional means. This transformation LLM is not (necessarily) the LLM under test, and is specified in the configuration file.

For tasks with multiple inputs (e.g., *premise* and *hypothesis* in NLI), if applicable, the relation will be applied to as many combination of inputs as possible. For instance, a relation could be applied to an NLI input's *premise*; to its *hypothesis*; or to both at once. This may result in multiple possible follow-up inputs from a single source input and MR.

Syntactic output comparisons may be done by direct equivalence, difference, set comparison, or another method, depending on the MR. Semantic output comparisons are done using cosine similarity via the BERT-based model PARAPHRASE-MINILM-L6-V2. We specify similarity thresholds to determine (depending on the MR) if two outputs are equivalent (0.8 by default) and different (0.4 by default). Numerical equivalence is determined by direct comparison with a 0.1 error window.

III. TOOL USAGE

Installation. LLMORPH requires PYTHON 3.10. Dependencies are installed via `python install -r requirements.txt`. An API key compatible with the OPENAI API format should be put into `security/token-key.jwt`. This key is used for both the LLM under test and the LLM-based MRs.

A. Running the tool

LLMORPH can be run using two methods: through a command line interface (CLI), or through a configuration file where we can set more parameters.

CLI. To run using CLI, execute `python src/main.py` with the the following arguments:

- `llm`: The ID of the LLM to test. This is the model name to be sent through the API.
- `task`: The name of the NLP task to test on. The list of tasks can be found in `src/config/list_tasks.json`.
- `mr`: The name of the metamorphic relation to test using. The list of relations can be found in `src/config/list_relations.json`.
- `input_data`: The path to the JSON file containing the inputs. Structured as an array of data points.
- `base_dir`: The path to the directory where caches and outputs will be stored.

Configuration file. To run using the config file (found at `src/config/run_config.json`), execute `python src/mt_main.py`. Some notable parameters include:

- `llm_list`: A list of LLM names. All LLMs in the list will be tested.
- `tasks`: A dictionary of lists, with task name as the key and a list of MRs as the value. An empty list will run all available MRs.
- `checkpoint_interval`: The number of instances processed before saving a checkpoint.
- `continue_from_checkpoint`: If `true`, will continue run from the checkpoint with the latest creation time.
- `llm_endpoint`: The endpoint for the LLM under test, as well as the LLM for MR implementation. Change if not using the default OpenAI endpoint.
- `llm_for_transformation`: The LLM used for MR implementation. Defaults to the LLM under test.

More parameter details can be found in `README.md`.

[2]A cirque is a geological feature that *forms* glaciers, not *formed by* them. The correct answer should be unknown

B. Reading the output

The results can be found in `{base_dir}/results` (with `base_dir` being specified in the run parameters). The output is a JSON file with a list of responses, with each instance specifying the following:

- `source_input`: The original test input.
- `source_output`: The response of the LLM from the source input.
- `followup_inputs`: A list containing the transformed input(s).
- `followup_outputs`: A list containing the responses of the LLM from the follow-up input(s).
- `relation`: A list specifying whether the follow-up output(s) satisfied the output relation.
- `verification_failure`: Whether the metamorphic group was valid according to any restrictions specified for the metamorphic relation.

C. Adding and modifying tasks, relations, and LLMs

Tasks. The currently implemented NLP tasks are done via prompts to the LLM. These prompts can be modified, and new tasks added, through editing `src/config/list_tasks.json` and `src/config/template/sut_prompt_templates.json`.

Metamorphic relations. The currently implemented MRs are done via either traditional function implementation, or prompts to an LLM. These can be modified, and new relations added, through editing the following: `src/relations/func_it.py` and `src/relations/func_or.py` for the implementation of the input transformation and output relation, respectively; `src/config/template/it_prompt_templates.json` or `src/config/template/or_prompt_templates.json` if using a prompted LLM for transformation or comparison; and `src/config/list_relations.json` to specify the particular MR.

LLMs. Currently, LLMORPH supports LLMs via compatibility with the OPENAI API format. This is controlled by the `llm_list` and `llm_endpoint` parameters in the config file. To use another API format, or to run a local model, modify `src/llm_runner.py`.

IV. EVALUATION

To evaluate the ability for LLMORPH to detect faulty behaviour, we conducted large-scale experiments using LLMORPH on three popular LLMs (GPT-4, LLAMA3, and HERMES 2) and four datasets (SQUAD2, SNLI, SST2, and RE-DOCRED), leading to 561,267 test executions. We found that LLMORPH effectively exposes faulty behaviours, with an average failure rate of 18%. Comparing with traditional testing with hard-to-obtain labelled data, we found it complementary, with MT being able to detect bugs not found by the former. We also manually analysed 937 metamorphic oracle violations and found a range of false positive rates, varying from 0% to 70%, depending on the MR and task. Most of these arise from the intrinsic limitations of MT for NLP and aligns with traditional MT for NLP [7]. In addition, we found that the evaluation of each input was fast, usually depending only on the speed of the 2-3 calls to the LLM per source input.

For more details (including on mitigating FPs and the efficacy of specific MRs), please refer to our ICSME paper [7].

V. RELATED WORK

Traditionally, testing of LLMs use benchmarks (e.g., MMLU [8]), where LLM outputs are compared to ground truths to determine effectiveness in the area of test. However, this requires labelled data, which is costly to obtain. LLMORPH instead utilises Metamorphic Testing, which does not require any labelled data, allowing cheaper fully automated testing.

Hyun et al. [6] recently presented METAL, a framework for MT for LLMs on NLP tasks. There are some limitations to their work, however, which we address in ours. They investigate 13 MRs, while we implement 36; they use zero-shot prompting in their LLM MR implementation, while we use few-shot; we have a CLI availability as well as a large number of configuration parameters, while they do not; our users can choose what MR to run, while theirs cannot; in ours it is easy to add new tasks and relations, while in theirs it is not; and ours is fully realised and modular, rather than a simple Jupyter Notebook.

VI. CONCLUSION AND FUTURE WORK

Despite the vast resources poured into producing and testing large language models, using Metamorphic Testing in this area is still unexpectedly underexplored. LLMORPH is one of the first to explore this space, providing a foundation from which to investigate MT for LLMs.

LLMORPH currently implements 36 out of the 191 Metamorphic Relations (MRs) we have collected [7]. By open-sourcing the tool and designing it to be modular and easily extensible, we hope the community will contribute to expanding its support for additional MRs and NLP tasks.

REFERENCES

[1] V. Terragni, A. Vella, P. Roop, and K. Blincoe, "The future of ai-driven software engineering," *ACM Transactions on Software Engineering Methodology (TOSEM)*, 2025.

[2] Y. Chang, X. Wang, J. Wang, Y. Wu, L. Yang, K. Zhu, H. Chen, X. Yi, C. Wang, Y. Wang *et al.*, "A survey on evaluation of large language models," *ACM Transactions on Intelligent Systems and Technology*, 2024.

[3] V. Hanke, T. Blanchard, F. Boenisch, I. E. Olatunji, M. Backes, and A. Dziedzic, "Open llms are necessary for current private adaptations and outperform their closed alternatives," *arXiv*, 2024.

[4] E. T. Barr, M. Harman, P. McMinn, M. Shahbaz, and S. Yoo, "The oracle problem in software testing: A survey," *IEEE transactions on software engineering*, vol. 41, no. 5, pp. 507–525, 2014.

[5] T. Y. Chen, F.-C. Kuo, H. Liu, P.-L. Poon, D. Towey, T. H. Tse, and Z. Q. Zhou, "Metamorphic testing: A review of challenges and opportunities," *ACM Computing Surveys*, vol. 51, no. 1, pp. 4:1–4:27, 2018.

[6] S. Hyun, M. Guo, and M. A. Babar, "Metal: Metamorphic testing framework for analyzing large-language model qualities," in *International Conference of Software Testing and Verification (ICST)*, 2024.

[7] S. Cho, S. Ruberto, and V. Terragni, "Metamorphic testing of large language models for natural language processing," in *International Conference on Software Maintenance and Evolution (ICSME)*, 2025.

[8] D. Hendrycks, C. Burns, S. Basart, A. Zou, M. Mazeika, D. Song, and J. Steinhardt, "Measuring massive multitask language understanding," *International Conference on Learning Representations (ICLR)*, 2021.