Variable Discovery with Large Language Models for Metamorphic Testing of Scientific Software

Christos Tsigkanos[1,2][0000−0002−9493−3404], Pooja Rani[3][0000−0001−5127−4042], Sebastian Müller[4][0000−0002−3057−1125], and Timo Kehrer[1][0000−0002−2582−5557]

[1]University of Bern, Switzerland
[2]University of Athens, Greece
[3]University of Zurich, Switzerland
[4]Humboldt-Universität zu Berlin, Germany

Abstract. When testing scientific software, it is often challenging or even impossible to craft a test oracle for checking whether the program under test produces the expected output when being executed on a given input – also known as the oracle problem in software engineering. Metamorphic testing mitigates the oracle problem by reasoning on necessary properties that a program under test should exhibit regarding multiple input and output variables. A general approach consists of extracting metamorphic relations from auxiliary artifacts such as user manuals or documentation, a strategy particularly fitting to testing scientific software. However, such software typically has large input-output spaces, and the fundamental prerequisite – extracting variables of interest – is an arduous and non-scalable process when performed manually. To this end, we devise a workflow around an autoregressive transformer-based Large Language Model (LLM) towards the extraction of variables from user manuals of scientific software. Our end-to-end approach, besides a prompt specification consisting of few examples by a human user, is fully automated, in contrast to current practice requiring human intervention. We showcase our LLM workflow over three case studies of scientific software documentation, and compare variables extracted to ground truth manually labelled by experts.

Keywords: Scientific Software · Metamorphic Testing · Large Language Models · Natural Language Processing

1 Introduction

With software being the most important driver for research in many scientific disciplines, the functional correctness of scientific software in particular is of utmost importance [8, 27, 28, 30]. Scientific software that has bugs may produce wrong outcomes leading to erroneous evidence, which in turn may have severe consequences in terms of research costs, scientific reputation, or even human well-being (e.g., when relying on invalid theories) [33, 42]. Software testing is the predominant way for ensuring functional correctness of a program under test. Traditional testing techniques rely on a test oracle for checking whether

ICCS Camera Ready Version 2023
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-35995-8_23

the program under test produces the expected output when being executed on a given input. In some cases, however, obtaining reliable oracles is challenging or even impossible – this is generally known as the test oracle problem in software engineering [4].

Scientific software is particularly affected by the oracle problem [23]. A major reason for this is that the scientific process is often exploratory in nature, with software written to find answers that are previously unknown [9, 47, 50], while lacking functional requirements being specified up-front [35]. Moreover, the underlying scientific theory may involve complex computations or inherent uncertainties, making it hard to determine the expected output for a given input [12, 26, 27, 50].

Metamorphic Testing [43] is a property-based software testing approach that mitigates the oracle problem by relying on so-called metamorphic relations [44]. Roughly speaking, a metamorphic relation specifies how the output changes according to a change made to the input. This way, a huge amount of test cases may be generated based on a *single* input-output-pair, invalidating the traditional prerequisite of being able to accurately determine the expected output for *any* given input of a program under test. By mitigating the oracle problem, metamorphic testing qualifies as a promising approach for testing scientific software [11, 12, 14, 21, 31, 32, 34]. However, scientific software is often characterized by having large input-output (I/O) spaces which are hard to cover by metamorphic relations when being specified ad-hoc, calling for methods to support scientific software engineers in systematically deriving metamorphic relations.

In recent work on metamorphic testing of scientific software, Peng et al. [36, 37] proposed to use auxiliary artifacts such as user manuals as a potential source for extracting metamorphic relations. The fundamental prerequisite consists of extracting input-output variables of interest; subsequently, relations can be devised with the overall goal of enabling metamorphic testing. The problem is illustrated in Fig. 1 over an excerpt of a scientific software manual: certain variables (e.g., "status", "startup") appear in the text, along with some hints on metamorphic relations (shaded in Fig. 1). Variable extraction has been initially performed manually [36], an arduous and non-scalable process. Following that, Peng et al. [37] proposed to semi-automate the variable extraction using supervised machine learning algorithms and manually crafted natural language processing-based patterns. However, the supervised learning methods still demand manual work in creating a ground truth, crafting the NLP features.

To further increase the level of automation, we devise a workflow around an autoregressive transformer-based Large Language Model (LLM) towards the first critical step of extracting I/O variables for metamorphic testing from documentation of scientific software. In contrast to manual extraction, our end-to-end approach is fully automated, besides a prompt specification consisting of few examples by a human user. We showcase variable extraction over documentation of scientific software and compare the variables extracted by our workflow to a ground truth that has been manually labelled by experts [37].

ICCS Camera Ready Version 2023
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-35995-8_23

We extend recent previous work [46] which appeared as a short paper at the Early Research Achievements track of the 30th IEEE International Conference on Software Analysis, Evolution and Reengineering, where we outlined the underlying research problem and illustrated emerging results. In this paper, we present our approach for variable discovery with LLMs for metamorphic testing in detail. We devote extensive discussion of evaluation aspects of the approach – investigating partial and exact discovered variables, performance of the implied binary classification test, as well as threats to validity. Specifically, our contributions are:

- We present a concrete instantiation of our approach for variable discovery from scientific software manuals for metamorphic testing with LLMs;
- We evaluate the advocated LLM workflow experimentally over three different case studies, demonstrating its feasibility and investigating its performance in tandem with operationalization options, and finally;
- We provide a replication package allowing our results to be reproduced by the research community.

The remainder of this paper is structured as follows. In Section 2 we contextualize our approach within the state of the art. In Section 3, we describe our solution in the form of a workflow revolving around an LLM, and discuss implementation particulars. Section 4 presents our evaluation and discusses results along with threats to validity. Finally, Section 5 concludes the paper and provides an outlook on future work.

2 State of the Art

A number of previous works has demonstrated the feasibility of metamorphic testing for testing scientific software, yet relying on metamorphic relations that have been manually specified in a largely ad-hoc manner [11, 12, 14, 21, 31, 32].

A major research stream on supporting the discovery of metamorphic relations is devoted to observing behavior of a running program. Su et al. [45] present KABU, a tool to automatically find metamorphic relations by generating new inputs for the program under test and then inferring relations in a rule-based manner. Kanewala et al. [17, 22, 24] investigate the applicability of different machine learning approaches on the task of metamorphic relation discovery. Hiremath et al. [18] apply machine learning to identify all possible metamorphic relations on oceanographic software that are then minimized according to a cost function. With these approaches, metamorphic relations are learned directly from the behavior of the program under test, which means that all found relations can only be used for regression testing of future program versions. Our approach is to learn metamorphic relations from auxiliary documents such as user manuals, and is thus more general.

In recent work on metamorphic testing of scientific software, Peng et al. [36, 37] proposed to use auxiliary artifacts such as user manuals, discussion forums, or documentation as a potential source for extracting metamorphic relations. As

ICCS Camera Ready Version 2023
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-35995-8_23

The on/off status$_\odot$ of pumps can be controlled dynamically by specifying startup$_\odot$ and shutoff water depths at the inlet node or through user-defined Control Rules. (...) For a Type 5 pump, its operating curve shifts position such that flow$_\odot$ changes in direct proportion to the controlled speed setting while head$_\odot$ changes in proportion to the setting squared. The principal input parameters for a pump include: =" names of its inlet and outlet$_\odot$ nodes = name of its pump curve (or * for an Ideal pump) = initial on/off status$_\odot$ = startup$_\odot$ and shutoff depths (optional). 3.2.9 Flow Regulators are structures or devices used to control and divert flows$_\odot$ within a conveyance system. They are typically used to: =" control releases from storage$_\odot$ facilities =" prevent unacceptable surcharging =" divert flow to treatment facilities and interceptors. SWMM can model the following types of Flow Regulators: Orifices$_\odot$, Weirs$_\odot$, and Outlets$_\odot$. Orifices$_\odot$ are used to model outlet and diversion structures in drainage systems (...) Orifices$_\odot$ can be used as storage$_\odot$ unit outlets$_\odot$ under all types of flow routing$_\odot$. If not attached to a storage unit node, they can only be used in drainage networks that are analyzed with Dynamic Wave flow$_\odot$ routing.

Fig. 1: Excerpt of a page from the Storm Water Management Model [41] scientific software manual by the U.S. Environmental Protection Agency, showing words that the LLM workflow correctly$_\odot$ classified as variables, and ones that it misclassified. Words not marked were correctly classified as non-variables. An instance of a metamorphic relation is shown shaded.

a first step towards metamorphic relation discovery from these artifacts, their goal is to identify input-output variables from natural language descriptions. Variable extraction has been initially performed manually [36], an arduous and non-scalable process; variables occurring throughout a scientific software manual were identified in a laborious task involving two researchers and amounting to 40 human-hours. Following that, Peng et al. [37] proposed to semi-automate the variable extraction using supervised machine learning algorithms and manually crafted natural language processing-based patterns. Although these methods are promising, they are limited as (i) they require considerable human interventions in preparing the ground truth and crafting features, and (ii) they rely on how similar variables are written in scientific software. Our approach aims at end-to-end automation and a more general applicability of extracting input-output variables.

Recent research has shown interest in exploring LLMs to overcome the general limitations of classical ML and NLP techniques. LLMs are trained on billions of parameters retrieved from large-scale natural language sources, e.g., web pages. Despite learning a specific task on a particular dataset, they have been shown to learn tasks without external supervision [38]. They can leverage learned features to work on a variety of other problems, such as in machine translation or spelling correction [7]. Specific to software engineering, LLMs have been explored e.g., for classifying issue reports [13], generating code from docstrings [6], generating docstrings from code [10], or synthesizing programs [3]. Given these results,

ICCS Camera Ready Version 2023
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-35995-8_23

we adopt LLMs towards our overall goal of automating MR discovery which has not yet been investigated for applications of LLMs in software engineering.

3 Discovering I/O Variables with an LLM

To extract I/O variables, we devise a data processing workflow revolving around a generative Large Language Model. Given a scientific software manual in PDF format, the target task consists of inferring the metamorphic variables inherent to it. Due to the underlying deep learning model being an LLM, special attention is given to prompt construction, discussed subsequently.

3.1 LLM-based Workflow

The data processing workflow we adopt for variable extraction is illustrated in Fig. 2 and revolves around an LLM and two stages: (i) pre-processing, where the scientific software manual is prepared to be submitted to the LLM, and (ii) post-processing, where I/O variables are extracted from its output. In the following, we detail key steps as components, noting that given a source document, all steps described are automated, except for a few human-specified examples used for the prompt.

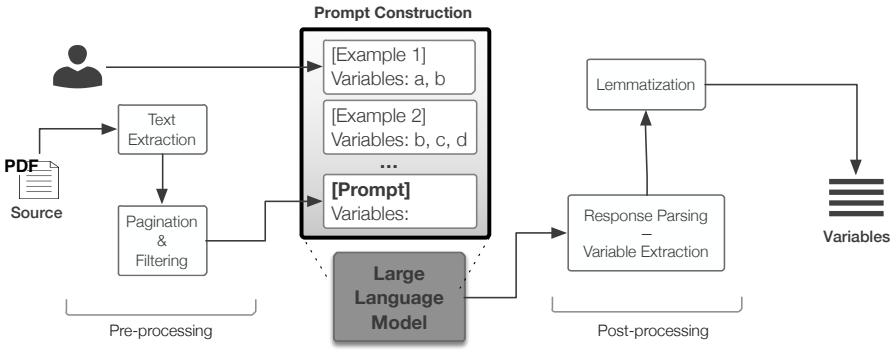


Fig. 2: Data analysis workflow adopted for inferring variables for metamorphic testing from a source document, with an LLM having the central role.

Text Extraction. This first step concerns extracting text from a source document, which is in PDF format, typical for scientific software documentation. Faithful to the automation objective pursued, no filtering by a human user is assumed. The input includes the *entirety* of the document including auxiliary content (e.g., tables of contents, title pages, etc.). For this initial step, to ensure that text is correctly extracted and PDF artifacts are kept minimal, we employ optical character recognition (OCR) on screenshots of pages – several options

ICCS Camera Ready Version 2023
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-35995-8_23

exist, e.g., tesseract and EasyOCR[1]. This is in contrast to extracting text directly from the PDF, which is prone to produce irregular results and depends strongly on the document encoding. We note that the realization of the OCR pipeline may affect the quality of text extracted.

Pagination and Filtering. The source text is subsequently filtered for irregular characters (e.g., special characters, OCR noise) and maintained in pagewise chunks according to the source document. At this step, pages with less than some specified number of characters can be omitted – such pages assumed to correspond to e.g., full-page figures, separator pages, etc., may not contain I/O variables.

LLM invocation. A prompt is constructed consisting of few-shot examples provided by the expert user (discussed subsequently in Sec. 3.2). Afterward, the prompt is tokenized according to the target model architecture and the LLM is subsequently invoked. Several autoregressive transformer-based LLMs can be utilized, open ones such as GPT-J-6B [49], GPT-NeoX-20B [5], or closed ones[2] such as OpenAI's GPT-3 or Google's PaLM.

Response Parsing & Variable Extraction. After invoking an inference operation over the prompt to the LLM, its textual response is parsed. The model in essence has completed the prompt given to it, by appending a list of potential variables. Those are extracted from the corresponding response and deduplicated.

Lemmatization. Observe that variables (as words), may occur in several forms in the source text, and as such they should be as uniquely identifiable as possible. This step refers to the process of turning a word into its lemma. A lemma is the "canonical form" of a word, commonly corresponding to its dictionary version – for instance, "flow rates" would be transformed to "flow rate". We perform such lemmatization for each variable extracted.

The result of the workflow is succinctly illustrated in the example of Fig. 1 over an excerpt of a page of the SWMM scientific manual [41]: certain words have been classified as variables, while others have been classified as not being variables. Furthermore, note that certain words may be misclassified – either as false positives (instances which are not relevant but which the model incorrectly identified as relevant), or false negatives (instances which are relevant but which the model incorrectly identified as not relevant).

3.2 Prompt Construction and LLM Particulars

A prompt for the LLM is compiled consisting of the few-shot labelled page examples provided by the user, along with the target page being processed appended to them – Fig. 3 illustrates a fragment of such a prompt, which starts with an instruction to the model, defining the task that it is required to do. The form of the prompt consists of labelled instances of "Text: $\mathcal{T}_i$ and Variables: $\mathcal{V}_i$"; where

[1] tesseract-ocr.github.io, github.com/JaidedAI/EasyOCR – uses ResNet, CTC, and beam-search-based decoder.

[2] openai.com/api, deepmind.com/publications, goo.gle/palm-paper

ICCS Camera Ready Version 2023
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-35995-8_23

$\mathcal{T}_i$ corresponds to the text of page i of the source document and $\mathcal{V}_i$ to a comma-separated list of variables occurring in page i, in order to steer the model to what output (and structure of the output) is expected. Subsequently, the text of the current page being processed is appended, and the prompt ends with an (empty) "Variables:" directive.

Your task is to extract Variables from the Text.
Text: (...) If no value for P_UPDIS is entered, the model will set P_UPDIS = 20. (...) NPERCO controls the amount of nitrate removed from the surface layer in runoff relative to the amount removed via percolation. The value of NPERCO can range from 0 to 1.0. (...) If no value for PHOSKD is entered, the model will set PHOSKD = 175.0. (...)
Variables: P_UPDIS, NPERCO, PPERCO, PHOSKD
Text: **[current page being processed]**
Variables:

Fig. 3: Fragment of the prompt given to the LLM, reflecting (part of) one example page labelled by the expert user, along with the current page being processed by the workflow. Note the initial instruction (first line) and that the prompt string ends with 'Variables:', that the LLM should complete by appending.

The response of few-shot LLMs can be unstable and be strongly dependent on the prompt format, the given specific examples, as well as their order. As such, prompt construction in LLMs plays a central role [40]. In our case, user input consists of few examples of text and certain words occurring in the text, which the expert user labels as variables. This user input provides a threat to *majority label bias*, where the model may suggest responses which are more frequent in the examples given by the user, throughout invocations. The intuition is that variable examples initially specified should be as descriptive, unique and prominent in the text as possible. However, as we discuss in future work, investigation into more effective prompt construction is a direction that warrants further consideration – a highly active topic in current LLM research. Since LLMs are sensitive to examples' ordering, we randomize the ones given by the user in each invocation (*recency bias*). *Common token bias*, where the model tends to yield tokens more common in its pre-training data, is an issue as well – this means that variables having more common names may be suggested with higher frequency.

Parameters common in LLM models are temperature, top-p, maximum length (in tokens, of the input and output), and frequency/presence penalties. Informally, lower temperature values render the model increasingly confident in its top choices, while higher decrease confidence while encouraging creative outputs. Top-p has a similar effect, while frequency/presence penalties can be used to suppress repetition.

ICCS Camera Ready Version 2023
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-35995-8_23

4 Evaluation

To concretely support evaluation and investigate feasibility, we realized a proof-of-concept implementation [1] reflecting the proposed workflow of Fig. 2. Thereupon, we assess our approach for variable inference. Specifically, we target its accuracy as a binary classifier (labelling words as variables or non-variables), as well as its performance as true positive rates. Subsequently, we discuss our findings, operationalization options and threats to validity.

4.1 Ground Truth and Experiment Setup

Experimental Subjects. Our evaluation target consists of three scientific software user manuals, in line with the state of the art: in foundational works in metamorphic testing [36, 37], I/O variables occurring throughout scientific software manuals were manually identified, in a laborious task. We acknowledge this significant effort and treat this result as the ground truth, against which we compare our LLM-based workflow.

SWMM. The Storm Water Management Model (SWMM [41]) by the U.S. Environmental Protection Agency (EPA), is a dynamic rainfall-runoff simulation software that computes runoff quantity and quality within mostly urban areas. The scientific users of SWMM include physicists, hydrologists and engineers involved in planning, analysis, and design related to storm water runoff, combined and sanitary sewers, and other drainage systems. Its user manual [41] is a 353-page PDF document; the ground truth [36] consists of 1005 I/O variables.

SWAT. The Soil and Water Assessment Tool (SWAT [2]) is a watershed to river basin-scale model used to simulate the quality and quantity of surface and groundwater as well as predicting the environmental impact of land use, land management practices, and climate change. SWAT is widely adopted to evaluate soil erosion prevention and control, non-point source pollution and overall regional management of watersheds. Its user manual [2] is a 649-page PDF document; the ground truth [37] consists of 1461 I/O variables.

MODFLOW. The Modular Hydrologic Model (MODFLOW [29]) targets groundwater-flow simulation, including groundwater/surface-water coupling, solute transport, land subsidence, and others. It is widely used for over 30 years by scientists, consultants and governmental organizations. Its user manual [29] is a 188-page PDF document; the ground truth [37] consists of 772 I/O variables.

Instantiation of the Workflow. We realized the workflow of Fig. 2 by employing the (medium-sized) GPT-J-6B [49] – an open-source transformer model trained using JAX [48], known to be trained on a mix of code and natural language text from several programming languages. Specifically, GPT-J-6B was trained on the Pile [15], a large-scale dataset curated by EleutherAI[3]. For our experiments, we deployed the workflow on an NVIDIA RTX A6000 (CUDA 12.0, PyTorch 1.13), over model GPT-J-6B (float32), with each invocation taking less

[3] www.eleuther.ai

ICCS Camera Ready Version 2023
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-35995-8_23

than 10 seconds. Regarding configuration, a few-shot prompt consisting of 3 examples of text – variable pairs is used at each invocation, followed by the source document page currently processed. For extracting text from the document, we employ EasyOCR. We ignore pages with less than 400 characters, to filter out pages with, e.g., full-sized figures, tables of contents, etc. Other than that, the respective document is provided to the processing pipeline page by page. We set presence penalty to 0.9, top-p to 0.7, and limit the number of new tokens to 35. Within each invocation, the order of the human-specified examples is randomized at the respective prompt. Thereupon, we invoke the workflow for different values of the critical temperature parameter. Variable discovery (excluding OCR) for the entirety of the source software manuals is in the range of 19 minutes for SWMM, 28 minutes for SWAT and 8 minutes for MODFLOW.

Evaluation Metrics. To assess our approach, we compare the workflow-extracted variables for each case study to the respective ground truth [36]. We first investigate *accuracy* of the approach, against the implied binary classification test. Subsequently, and for a more refined metric, we assess *performance* achieved as true positives over different values of the critical temperature parameter.

Accuracy. We evaluate the advocated LLM workflow as a binary classifier, where the task is to classify every word appearing in the source document as a variable – or not a variable. Following previous works in this direction [16,20,39], we compute accuracy to evaluate our results. For a binary classifier, recall that accuracy is defined as the proportion of correct predictions (both true positives and true negatives) among the total number of cases examined. To this end, we compare the output of the workflow given the source document against ground truth; words of the source PDF are defined as unique instances of characters occurring between spaces, which are subsequently lemmatized.

True Positive Rate. Recall the example of Fig. 1, which illustrates the workflow's outputs for an example page fragment. We consider two types of *true positives*: (i) exact true positives, where the LLM workflow derived variable exactly matches one in ground truth, and (ii) partial true positives, where the workflow derived a variable which is part of one in ground truth. The latter is because variables are often phrasal, and the LLM workflow may label part of the phrase – for example, derived variable "water depth" partially occurs within "maximum water depth" specified in ground truth. We treat as partial, (standalone) variables identified by the workflow which are comprised of at least 2 characters, occurring within a ground truth variable. We specifically investigate partial variables due to the high potential they have to be integrated within an interactive human-machine labelling process.

4.2 Results

Experiment results are summarized in Tab. 1. We obtain a best accuracy of 0.88 (SWMM case), 0.91 (SWAT case) and 0.90 (MODFLOW case). Accuracy, up to the second digit is the same for both partial and exact matches. Table 1 further shows in detail the number of true positives for exact and partial matches and

ICCS Camera Ready Version 2023
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-35995-8_23

accuracy achieved over the unique lemmatized words of the source document, for best and worst temperature values.

	Temp	True P. (exact)	True P. (partial)	Unique Lemmas	Accuracy
SWMM ⋆	0.4	250	415	11154	0.88
SWMM	0.9	228	403	11154	0.87
SWAT ⋆	0.2	692	1019	18013	0.91
SWAT	0.9	672	992	18013	0.91
MODFLOW ⋆	0.7	223	271	4874	0.90
MODFLOW	0.9	220	263	4874	0.90

Table 1: Accuracy of the LLM workflow as a binary classifier. Illustrated is accuracy achieved for best (⋆) and worst temperature for exact variable matches.

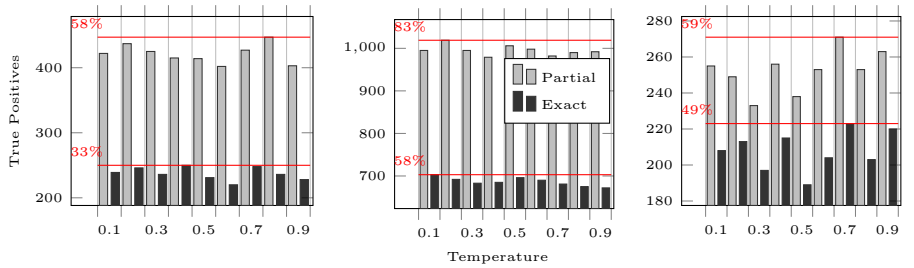
Observe that the critical temperature parameter does not significantly affect accuracy – as such, we subsequently investigate performance as manifested in the true positive rate. Specifically, we assess performance as true positives over the ground truth – that is, variables correctly identified against ground truth – for different temperature parameters, and for both exact and partial matches. Our results are illustrated in Fig. 4. Notably, the workflow successfully derives 58% (SWMM), 83% (SWAT) and 59% of the ground truth as partial matches, and 33% (SWMM), 58% (SWAT) and 49% (MODFLOW) as exact matches. Overall results imply a certain trend: moderate to medium temperature values yield the best true positive rate (Fig. 4), while deviation in accuracy between best and worst (Tab. 1) remains relatively low.

Conversely, Fig. 5 illustrates false positives, as the number of variables discovered for each case which are not relevant and which the model incorrectly identified as relevant. False positive rate generally increases with temperature – behavior in line with the definition of the temperature parameter, where higher values render the LLM more creative with its choices. Notably, for each case, there is a (low) temperature value with which false positives are (relatively) low.

4.3 Discussion and Threats to Validity

We especially note that besides a few examples given for the prompt, this is the output of an automatic procedure utilizing a model not fine-tuned; we believe it illustrates significant future potential and warrants further investigation. However, we acknowledge that due to the generic generative model used, there is a high amount of false positives (Fig. 5), especially against tailored approaches [36]. The juxtaposition of Figures 5 and 4 shows that a balance between true positives and false positives needs to be achieved regarding the choice of temperature. Fine-tuning the LLM is the next major conceptual direction to investigate to mitigate this.

ICCS Camera Ready Version 2023
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-35995-8_23



(a) SWMM (b) SWAT (c) MODFLOW

Fig. 4: Overall performance for the three case studies considered as true positives correctly derived by the advocated LLM workflow against ground truth. Illustrated over different temperature parameters, for variables partially identified (gray bars) and exactly identified (black bars). The maximum percentage of ground truth correctly identified in both cases is also marked.

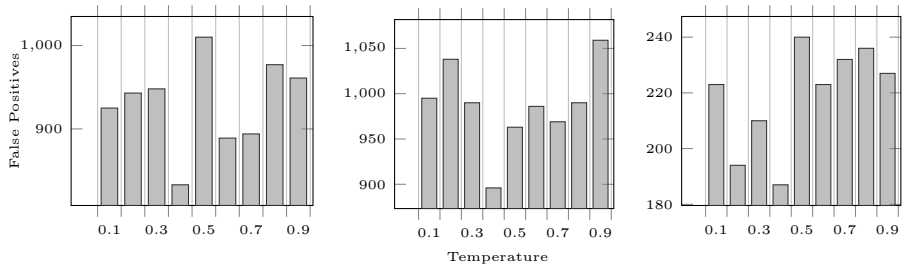
Regarding operationalization, an invocation of the workflow against a page of the source document takes seconds, rendering the approach fitting to an application where a user interacts with the workflow in an online fashion. Such a human-in-the-loop approach can target a trade-off between manual labelling and full automation. While leveraging partial variable matches for this represents an obvious way forward, one can envision the human expert user correcting model output, especially regarding false positives, and steering the model towards better overall task performance.

Threats to validity of our investigation revolve around the use of the generative LLM, which is the pillar of our approach. LLMs, by design, are trained against huge corpuses of general text [15] presenting a threat to construct validity. Regarding external validity, we note the difference of performance between, e.g., the SWAT case and the MODFLOW case. The conjecture is that natural language, language structure, and variable name choice inherent in a specific document against the LLM (and its training) leads to varying performance. As such, our results may not generalize to other scientific software documentation, although we performed 3 case studies. Additionally, we treated the scientific software document page by page; this represents a first approach, and making use of context of particular text (e.g., same paragraph, same section) is likely to lead to better results. As for threats to conclusion validity and repeatability, we release our analysis data and implementation in the form of a reproduction kit [1].

5 Conclusion and Future Work

Metamorphic testing involves reasoning on necessary properties that a program under test should exhibit regarding multiple input and output variables. A general approach consists of extracting metamorphic relations from auxiliary ar-

ICCS Camera Ready Version 2023
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-35995-8_23



(a) SWMM (b) SWAT (c) MODFLOW

Fig. 5: False positive rate illustrated over different temperature parameters.

tifacts such as user manuals or documentation, a strategy particularly fitting to testing scientific software. However, large input-output spaces are common and relations can be complex, hindering the application of metamorphic testing. The fundamental prerequisite consists of extracting input-output variables of interest; subsequently, relations can be devised with the overall goal of enabling testing. By virtue of our workflow design and the preliminary results presented, we believe to have demonstrated the potential that LLMs have for this critical step of variable discovery. Thereupon, we identify key research directions towards model refinement, prompt construction, and human-in-the-loop configurations.

Firstly, larger and more advanced models are very likely to perform better [25]. Naturally, we identify model fine-tuning as the key driver for achieving higher performance [19]. A systematic investigation into more effective prompt construction is warranted, especially towards issues raised by recency bias, common token bias, and majority label bias is a priority (as occurring in the particular problem tackled). Optimizations can include mixing previous model responses in the few-shot prompt, along with the one that is human-specified. This can steer the model into the current context, by providing more information about the topics inherent in neighboring text. Additionally, instead of autoregressive language models, Masked Language Models (MLMs [51]) can be employed. MLMs can be used to predict masked text parts based on its neighboring context [51]. Finally, leveraging partial variable matches represents an obvious way forward for operationalization; one can envision the human expert user correcting model output, especially regarding false positives, and steering the model towards better overall task performance.

Acknowledgements Funded in part by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) – SFB 1404 FONDA.

References

1. Implementation of the LLM-based workflow and reproduction kit. https://seg.inf.unibe.ch/papers/mt-varextract-gpt-0.7.tar.gz, 2023.

ICCS Camera Ready Version 2023
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-35995-8_23

2. J.G. Arnold, J.R. Kiniry, R. Srinivasan, J.R. Williams, E.B. Haney, and S.L. Neitsch. United States Department of Agriculture. Soil and Water Assessment Tool (SWAT). Texas Water Resources Institute, 2012.
3. Matej Balog, Alexander L Gaunt, Marc Brockschmidt, Sebastian Nowozin, and Daniel Tarlow. Deepcoder: Learning to write programs. arXiv preprint arXiv:1611.01989, 2016.
4. Earl T Barr, Mark Harman, Phil McMinn, Muzammil Shahbaz, and Shin Yoo. The oracle problem in software testing: A survey. IEEE transactions on software engineering, 41(5):507–525, 2014.
5. Sid Black et al. Gpt-neox-20b: An open-source autoregressive language model. arXiv preprint arXiv:2204.06745, 2022.
6. Tom Brown et al. Language models are few-shot learners. Advances in neural information processing systems, 33:1877–1901, 2020.
7. Andrew Carlson and Ian Fette. Memory-based context-sensitive spelling correction at web scale. In Intl. Conf. on Machine Learning and Applications, pages 166–171. IEEE, 2007.
8. Jeffrey C Carver, Neil P Chue Hong, and George K Thiruvathukal. Software engineering for science. CRC Press, 2016.
9. Jeffrey C Carver, Richard P Kendall, Susan E Squires, and Douglass E Post. Software development environments for scientific and engineering software: A series of case studies. In Intl. Conf. on Software Engineering, pages 550–559. IEEE, 2007.
10. Mark Chen et al. Evaluating large language models trained on code. arXiv:2107.03374, 2021.
11. Tsong Yueh Chen, Jianqiang Feng, and TH Tse. Metamorphic testing of programs on partial differential equations: a case study. In Intl. Computer Software and Applications, pages 327–333. IEEE, 2002.
12. Tsong Yueh Chen, Joshua WK Ho, Huai Liu, and Xiaoyuan Xie. An innovative approach for testing bioinformatics programs using metamorphic testing. BMC bioinformatics, 10(1):1–12, 2009.
13. Giuseppe Colavito, Filippo Lanubile, and Nicole Novielli. Issue report classification using pre-trained language models. In Intl. Workshop on Natural Language-Based Software Engineering, pages 29–32, 2022.
14. Junhua Ding, Dongmei Zhang, and Xin-Hua Hu. An application of metamorphic testing for testing scientific software. In Intl. Workshop on Metamorphic Testing, pages 37–43, 2016.
15. Leo Gao et al. The pile: An 800gb dataset of diverse text for language modeling. arXiv preprint arXiv:2101.00027, 2020.
16. Chengcheng Han, Zeqiu Fan, Dongxiang Zhang, Minghui Qiu, Ming Gao, and Aoying Zhou. Meta-learning adversarial domain adaptation network for few-shot text classification. arXiv preprint arXiv:2107.12262, 2021.
17. Bonnie Hardin and Upulee Kanewala. Using semi-supervised learning for predicting metamorphic relations. In Intl. Workshop on Metamorphic Testing, pages 14–17. IEEE, 2018.
18. Dilip J Hiremath, Martin Claus, Wilhelm Hasselbring, and Willi Rath. Towards automated metamorphic test identification for ocean system models. In Intl. Workshop on Metamorphic Testing, pages 42–46. IEEE, 2021.
19. Jeremy Howard and Sebastian Ruder. Universal language model fine-tuning for text classification, 2018.
20. Xiang Jiang, Mohammad Havaei, Gabriel Chartrand, Hassan Chouaib, Thomas Vincent, Andrew Jesson, Nicolas Chapados, and Stan Matwin. On the impor-

ICCS Camera Ready Version 2023
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-35995-8_23

tance of attention in meta-learning for few-shot text classification. arXiv preprint arXiv:1806.00852, 2018.
21. Upulee Kanewala and James M Bieman. Techniques for testing scientific programs without an oracle. In Intl. Workshop on Software Engineering for Computational Science and Engineering, pages 48–57. IEEE, 2013.
22. Upulee Kanewala and James M Bieman. Using machine learning techniques to detect metamorphic relations for programs without test oracles. In Intl. Symposium on Software Reliability Engineering, pages 1–10. IEEE, 2013.
23. Upulee Kanewala and James M Bieman. Testing scientific software: A systematic literature review. Information and software technology, 56(10):1219–1232, 2014.
24. Upulee Kanewala, James M Bieman, and Asa Ben-Hur. Predicting metamorphic relations for testing scientific software: a machine learning approach using graph kernels. Software testing, verification and reliability, 26(3):245–269, 2016.
25. Jared Kaplan, Sam McCandlish, Tom Henighan, Tom B Brown, Benjamin Chess, Rewon Child, Scott Gray, Alec Radford, Jeffrey Wu, and Dario Amodei. Scaling laws for neural language models. arXiv preprint arXiv:2001.08361, 2020.
26. Diane Kelly and Rebecca Sanders. The challenge of testing scientific software. In Annual Conference of the Association for Software Testing, pages 30–36, 2008.
27. Diane Kelly, Rebecca Sanders, et al. Assessing the quality of scientific software. In Intl. Workshop on Software Engineering for Computational Science and Engineering, 2008.
28. Diane Kelly, Spencer Smith, and Nicholas Meng. Software engineering for scientists. Computing in Science & Engineering, 13(05):7–11, 2011.
29. CD Langevin, JD Hughes, ER Banta, AM Provost, RG Niswonger, and Sorab Panday. Modflow 6 modular hydrologic model version 6.2. 1: US geological survey software release, 18 february 2021 https://doi. org/10.5066, 2021.
30. Ulf Leser, Marcus Hilbrich, Claudia Draxl, Peter Eisert, Lars Grunske, Patrick Hostert, Dagmar Kainmüller, Odej Kao, Birte Kehr, Timo Kehrer, Christoph Koch, Volker Markl, Henning Meyerhenke, Tilmann Rabl, Alexander Reinefeld, Knut Reinert, Kerstin Ritter, Björn Scheuermann, Florian Schintke, Nicole Schweikardt, and Matthias Weidlich. The Collaborative Research Center FONDA. Datenbank-Spektrum, (1610-1995), November 2021.
31. Xuanyi Lin, Michelle Simon, and Nan Niu. Exploratory metamorphic testing for scientific software. Computing in science & engineering, 22(2):78–87, 2018.
32. Xuanyi Lin, Michelle Simon, and Nan Niu. Hierarchical metamorphic relations for testing scientific software. In Intl. Workshop on Software Engineering for Science, pages 1–8, 2018.
33. Greg Miller. A scientist's nightmare: Software problem leads to five retractions. Science, 314(5807):1856–1857, 2006.
34. Sebastian Müller, Valentin Gogoll, Anh Duc Vu, Timo Kehrer, and Lars Grunske. Automatically finding metamorphic relations in computational material science parsers. In Intl. Workshop on Software Engineering for eScience, 2022.
35. Luke Nguyen-Hoan, Shayne Flint, and Ramesh Sankaranarayana. A survey of scientific software development. In Intl. symposium on empirical software engineering and measurement, pages 1–10, 2010.
36. Zedong Peng, Xuanyi Lin, Nan Niu, and Omar I Abdul-Aziz. I/O associations in scientific software: A study of SWMM. In Intl. Conf. on Computational Science, pages 375–389. Springer, 2021.
37. Zedong Peng, Xuanyi Lin, Sreelekhaa Nagamalli Santhoshkumar, Nan Niu, and Upulee Kanewala. Learning I/O variables from scientific software's user manuals. In Intl. Conf. on Computational Science, pages 503–516. Springer, 2022.

ICCS Camera Ready Version 2023
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-35995-8_23

38. Alec Radford et al. Language models are unsupervised multitask learners. OpenAI blog, 1(8):9, 2019.
39. Xiang Ren, Wenqi He, Meng Qu, Lifu Huang, Heng Ji, and Jiawei Han. Afet: Automatic fine-grained entity typing by hierarchical partial-label embedding. In Conf. on empirical methods in natural language processing, pages 1369–1378, 2016.
40. Laria Reynolds and Kyle McDonell. Prompt programming for large language models: Beyond the few-shot paradigm. In Extended Abstracts, CHI, pages 1–7, 2021.
41. Lewis A Rossman. Storm water management model user's manual, version 5.0. Cincinnati: National Risk Management Research Laboratory, Office of Research and Development, US Environmental Protection Agency, 2010.
42. Rebecca Sanders and Diane Kelly. Dealing with risk in scientific software development. IEEE software, 25(4):21–28, 2008.
43. Sergio Segura, Gordon Fraser, Ana B Sanchez, and Antonio Ruiz-Cortés. A survey on metamorphic testing. IEEE Transactions on software engineering, 42(9):805–824, 2016.
44. Sergio Segura, Dave Towey, Zhi Quan Zhou, and Tsong Yueh Chen. Metamorphic testing: Testing the untestable. IEEE Software, 37(3):46–53, 2018.
45. Fang-Hsiang Su, Jonathan Bell, Christian Murphy, and Gail Kaiser. Dynamic inference of likely metamorphic properties to support differential testing. In Intl. Workshop on Automation of Software Test, pages 55–59. IEEE, 2015.
46. Christos Tsigkanos, Pooja Rani, Sebastian Müller, and Timo Kehrer. Large language models: The next frontier for variable discovery within metamorphic testing? In International Conference on Software Analysis, Evolution and Reengineering, Early Research Achievements (ERA) track. IEEE Computer Society, 2023.
47. Anh Duc Vu, Timo Kehrer, and Christos Tsigkanos. Outcome-preserving input reduction for scientific data analysis workflows. In 37th IEEE/ACM International Conference on Automated Software Engineering, ASE 2022, Rochester, MI, USA, October 10-14, 2022, pages 182:1–182:5. ACM, 2022.
48. Ben Wang. Mesh-Transformer-JAX: Model-Parallel Implementation of Transformer Language Model with JAX. `github.com/kingoflolz/mesh-transformer-jax`.
49. Ben Wang and Aran Komatsuzaki. GPT-J-6B: A 6 Billion Parameter Autoregressive Language Model. `github.com/kingoflolz/mesh-transformer-jax`, 2022.
50. Elaine J Weyuker. On testing non-testable programs. The Computer Journal, 25(4):465–470, 1982.
51. Frank F Xu, Uri Alon, Graham Neubig, and Vincent Josua Hellendoorn. A systematic evaluation of large language models of code. In ACM SIGPLAN Intl. Symposium on Machine Programming, pages 1–10, 2022.

ICCS Camera Ready Version 2023
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-35995-8_23