2023 IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER) | 978-1-6654-5278-6/23/$31.00 ©2023 IEEE | DOI: 10.1109/SANER56733.2023.00070

Large Language Models: The Next Frontier for Variable Discovery within Metamorphic Testing?

Christos Tsigkanos[∞], Pooja Rani[+], Sebastian Müller[*] and Timo Kehrer[∞]
[∞]University of Bern, Switzerland
[+]University of Zurich, Switzerland
[*]Humboldt-Universität zu Berlin, Germany

***Abstract*—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 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-shot examples by a human user, is fully automated, in contrast to current practice requiring human intervention. We showcase our LLM workflow over a real case, and compare variables extracted to ground truth manually labelled by experts. Our preliminary results show that our LLM-based workflow achieves an accuracy of 0.87, while successfully deriving 61.8% of variables as partial matches and 34.7% as exact matches.**

***Index Terms*—Metamorphic Testing, Large Language Models, Natural Language Processing, Scientific Software**

I. Introduction

Metamorphic testing [9] is a property-based software testing technique involving metamorphic relations – those are transformations on the input and output of a program under test, that are able to predict the relative changes of output pairs given the corresponding input pairs [28]. Thus, metamorphic testing mitigates the test oracle problem [3] and enables test case generation. A major research stream on discovering metamorphic relations is devoted to observing behavior of a running program [13], [17], [18], [29], [20], a strategy however limited to its usefulness for regression testing. A much more general approach is to extract relations from auxiliary artifacts such as user manuals, discussion forums, or documentation. This strategy particularly fits to testing scientific software, which heavily suffers from the lack of test oracles [7], [30]. However, large input-output (I/O) spaces are common and relations can be complex, hindering the application of metamorphic testing.

The fundamental prerequisite of reasoning with metamorphic relations consists of extracting input-output variables of interest; subsequently, relations can be devised with the overall goal of enabling testing. The problem is illustrated in Fig. 1 over an excerpt of a scientific manual: certain variables (e.g., "status", "startup") appear in the text. Observe also that a metamorphic relation also appears (shaded in Fig. 1). Variable extraction has been initially performed manually, an arduous and non-scalable process; in foundational work [21], 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. proposed using supervised machine learning algorithms and manually crafted natural language processing-based patterns [22]. However, the supervised learning methods still demand manual work in creating a ground truth, crafting the NLP features.

To this end, we devise a workflow around an autoregressive transformer-based Large Language Model (LLM) towards the first critical step of extracting variables for metamorphic testing from user manuals of scientific software. In contrast to manual extraction, our end-to-end approach is fully automated, besides a prompt specification consisting of few-shot examples by a human user. We showcase variable extraction over the manual of the Storm Water Management Model (SWMM [26]) by the U.S. Environmental Protection Agency and compare the variables extracted by our workflow to ground truth previously [21] manually labeled by experts. Our preliminary results show that when considered as a binary classifier, our LLM-based workflow achieves an accuracy of 0.87, while successfully deriving 61.8% of the ground truth variables as partial matches and 34.7% as exact matches. We conclude the paper with an outlook on a future research agenda.

II. 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 text from a 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 outline key steps as components, noting that given a source document, all steps described are automated, except for few human-specified examples used for the prompt. Special attention is given to prompt construction and LLM parametrization, discussed subsequently.

Funded in part by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) – Project-ID 414984028 – SFB 1404 FONDA.

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 SWMM software manual, 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. *Workflow Components*

Text Extraction. This first step concerns extracting text from a source document in PDF format, as is typical for scientific software manuals. 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 OCR on screenshots of pages. This is in contrast to extracting text directly from the PDF, which is prone to produce irregular results depending on the document encoding.

Pagination and Filtering. The source text is subsequently filtered for irregular characters (e.g., special characters, OCR noise) and maintained into page chunks according to the source document. Pages with less than some specified number of characters are omitted (assumed to correspond to e.g., full-page figures, separator pages, etc.).

LLM invocation. A prompt for the LLM is constructed consisting of the few-shot labeled page examples by the user, along with the target page being processed appended to them. The form of such a constructed prompt consists of several instances of "Text: $\mathcal{T}_i$ and Variables: $\mathcal{V}_i$"; where $\mathcal{T}_i$ 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) is expected. Subsequently, the prompt is tokenized according to the model architecture and the LLM is subsequently invoked.

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 variables. Those are extracted from the corresponding response and deduplicated.

Lemmatization. Observe that variables (as words), may occur in several forms, 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, usually 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 [26]: certain words have been classified as variables, while others have been classified as being not variables. Furthermore, note that certain words may be misclassified – either as false positives, or false negatives.

B. *Prompt Construction and LLM Parametrization*

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 [25]. 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.

III. Preliminary Results

To concretely support evaluation and investigate feasibility, we realized a proof-of-concept implementation reflecting the proposed workflow of Fig. 2. Thereupon, we assess our approach for variable inference. Specifically, we target its accuracy as well as its performance as true positive rates.

Ground Truth. The Storm Water Management Model (SWMM [26]) 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

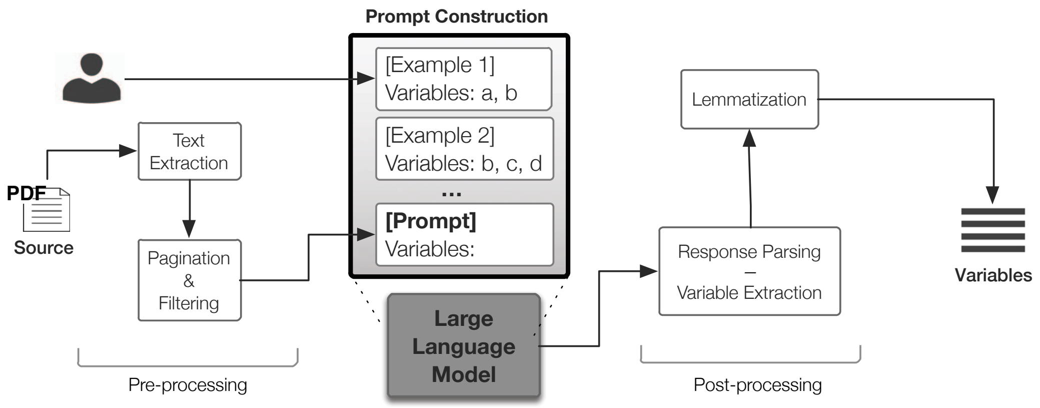

Fig. 2. Data analysis workflow adopted for inferring variables for metamorphic testing from a source document, with an LLM having the central role.

sewers, and other drainage systems. Its user manual [26] is a 353-page PDF document. In foundational work in metamorphic testing [21], I/O variables occurring throughout the SWMM document were manually identified, in a laborious task involving two researchers and amounting to 40 human-hours. 971 I/O variables were manually labelled; we treat this result as the ground truth, against which we compare our LLM workflow. For our experiments' prompt construction, we use the two examples specified in Fig. 3 of the original paper [21] as well as another specified by the authors – 3 in total.

Configuration and deployment. We realized the workflow of Fig. 2 by deploying the (medium-sized) GPT-J-6B [32] – an open-source transformer model trained using JAX [31], 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 [11], a large-scale curated dataset by EleutherAI[1]. We employ CPU inference on a VM with 64 GB of RAM on a libvirt host with an AMD Ryzen 7 3800X, with each invocation taking about 2 minutes. Regarding configuration, a few-shot prompt consisting of the 3 examples of text – variable pairs is used at each invocation, followed by the source document page currently processed. We ignore pages with less than 500 characters, to filter out pages with, e.g., full-sized figures, tables of contents, etc. Other than that, the document is provided to the processing pipeline as-is, page by page. We set presence penalty to 0.9, top-p to 0.8 and limit the number of new tokens to 35. Within each invocation, 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.

Results. To assess our approach, we compare the workflow-extracted variables to ground truth [21]. 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. Recall the example of Fig. 1, which illustrates workflow's outputs for a page. 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.

[1]www.eleuther.ai

TABLE I
ACCURACY OF THE LLM WORKFLOW AS A BINARY CLASSIFIER.

	True P. (exact)	True P. (partial)	Unique Lemmas	Accuracy Metric
Temp: 0.1	221	398	11027	0.86
Temp: 0.2	243	427		0.86
Temp: 0.3	223	403		0.86
Temp: 0.4	261	439		0.87
Temp: 0.5	215	385		0.86
Temp: 0.6	233	427		0.86
Temp: 0.7	222	401		0.86
Temp: 0.8	233	406		0.86
Temp: 0.9	**267**	**449**		**0.87**
Temp: 1.0	226	409		0.86

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 [24], [16], [12], 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. We obtain a best accuracy of 0.87 with temperature 0.9 (for both partial and exact matches). Table I shows in detail true positives for exact and partial matches and accuracy achieved over the unique lemmatized words, for different temperature values. Observe that the critical temperature parameter does not significantly affect accuracy – as such, we subsequently investigate performance, as manifested in the form of 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. 3. Notably, the workflow successfully derives 61.85% of the ground truth as partial matches and 34.77% as exact matches with a (best) temperature parameter of 0.9. Recall that high temperature values encourage more creative outputs. We especially note that this is the output of an automatic procedure which is not at all model fine-tuned; we believe it illustrates significant future potential and warrants further investigation.

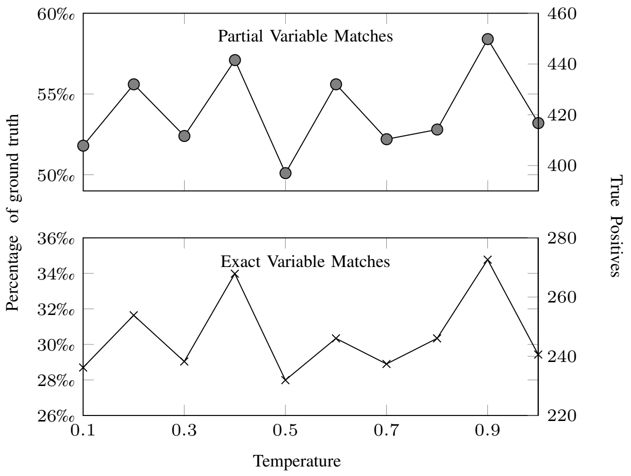

Fig. 3. Overall performance as percentage of SWMM ground truth (left axis) and corresponding true positives (right axis) correctly derived by the advocated LLM workflow, over different temperature parameters, for variables partially identified (top) and exactly identified (bottom).

IV. State of the Art

A recurring theme in software testing research is how to automate it, as manual testing is a time-consuming and expensive task. This is true for metamorphic testing (MT) as well, in particular for automating metamorphic relation (MR) discovery. Su et al. [29] present KABU, a tool to automatically find MRs by generating new inputs for the program under test and then inferring relations in a rule-based manner. Kanewala et al. [13], [17], [18] investigate the applicability of different machine learning approaches on the task of MR discovery. Hiremath et al. [14] apply machine learning to identify all possible MRs on oceanographic software that are then minimized according to a cost function. Both here and in Kanewala's approaches, and similarly to KABU, however, MRs 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.

As a first step towards MR discovery from auxiliary documents, several approaches have been suggested to identify input-output variables from manuals and forums. Those vary from manual identification [21] to using information retrieval methods based on ML and NLP techniques [22]. 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 [23]. They can leverage learned features to work on a variety of other problems, such as in spelling correction [6] or machine translation [1]. Specific to software engineering, LLMs have been explored for classifying issue reports [10], detecting and correcting spelling errors [27], generating code from docstrings [5], generating docstrings from code [8], or synthesizing programs [2]. Given these results, we adopt LLMs towards an overall goal of automating MR discovery.

V. Challenges and Research Agenda

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 artifacts 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 (i) model refinement, (ii) prompt construction, and (iii) human-in-the-loop configurations.

Firstly, larger models such as GPT-NeoX-20B [4], the upcoming BigScience's BLOOM or closed ones[2] such as OpenAI's GPT-3 or Google's PaLM are very likely to perform better [19]. In our investigation, we opted for an open model, faithful to the open scientific spirit. Naturally, we identify model fine-tuning or re-training as the key drivers for achieving higher performance. We especially note that besides the examples on the few-shot prompt which are human-specified at the beginning of the process, the entirety of the process is automatic. Furthermore, a systematic investigation into more effective prompt construction is warranted, especially towards issues raised by recency bias, common token bias, and majority label bias – as occurring in the particular problem tackled – is a priority. The effect of key model parameters on false positives and false negatives beyond temperature

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

(i.e., related top-p, frequency and presence penallties) should be quantitatively and systematically assessed. 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 neighbouring text.

Nevertheless, the major direction for improvement is fine-tuning [15] the underlying language model. Additionally, instead of autoregressive language models, Masked Language Models (MLMs [33]) can be employed for identifying variables from user manuals. Autoregressive models treat the token sequence left to right, predicting the next token based on previous ones, as such they can be less suitable for classification tasks where it is required to interpret the whole sequence. In such cases, MLMs can be used to predict masked text parts based on its neighbouring context [33].

Finally and with a broader view, human-in-the-loop approaches 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.

References

[1] Bahdanau, D., Cho, K., Bengio, Y.: Neural machine translation by jointly learning to align and translate. arXiv preprint arXiv:1409.0473 (2014)

[2] Balog, M., Gaunt, A.L., Brockschmidt, M., Nowozin, S., Tarlow, D.: Deepcoder: Learning to write programs. arXiv preprint arXiv:1611.01989 (2016)

[3] Barr, E.T., et al: The oracle problem in software testing: A survey. IEEE transactions on software engineering **41**(5), 507–525 (2014)

[4] Black, S., Biderman, S., Hallahan, E., Anthony, Q., Gao, L., Golding, L., He, H., Leahy, C., McDonell, K., Phang, J., et al.: Gpt-neox-20b: An open-source autoregressive language model. arXiv preprint arXiv:2204.06745 (2022)

[5] Brown, T., et al: Language models are few-shot learners. Advances in neural information processing systems **33**, 1877–1901 (2020)

[6] Carlson, A., Fette, I.: Memory-based context-sensitive spelling correction at web scale. In: Sixth International Conference on Machine Learning and Applications (ICMLA 2007). pp. 166–171. IEEE (2007)

[7] Carver, J.C., Hong, N.P.C., Thiruvathukal, G.K.: Software engineering for science. CRC Press (2016)

[8] Chen, M., Tworek, J., Jun, H., Yuan, Q., Pinto, H.P.d.O., Kaplan, J., Edwards, H., Burda, Y., Joseph, N., Brockman, G., et al.: Evaluating large language models trained on code. arXiv:2107.03374 (2021)

[9] Chen, T.Y., Cheung, S.C., Yiu, S.: Metamorphic testing: A new approach for generating next test cases. abs/2002.12543. CoRR (2020)

[10] Colavito, G., Lanubile, F., Novielli, N.: Issue report classification using pre-trained language models. In: 2022 IEEE/ACM 1st International Workshop on Natural Language-Based Software Engineering (NLBSE). pp. 29–32 (2022)

[11] Gao, L., Biderman, S., Black, S., Golding, L., Hoppe, T., Foster, C., Phang, J., He, H., Thite, A., Nabeshima, N., et al.: The pile: An 800gb dataset of diverse text for language modeling. arXiv preprint arXiv:2101.00027 (2020)

[12] Han, C., Fan, Z., Zhang, D., Qiu, M., Gao, M., Zhou, A.: Meta-learning adversarial domain adaptation network for few-shot text classification. arXiv preprint arXiv:2107.12262 (2021)

[13] Hardin, B., Kanewala, U.: Using semi-supervised learning for predicting metamorphic relations. In: 2018 IEEE/ACM 3rd International Workshop on Metamorphic Testing (MET). pp. 14–17. IEEE (2018)

[14] Hiremath, D.J., Claus, M., Hasselbring, W., Rath, W.: Towards automated metamorphic test identification for ocean system models. In: 2021 IEEE/ACM 6th International Workshop on Metamorphic Testing (MET). pp. 42–46. IEEE (2021)

[15] Howard, J., Ruder, S.: Universal language model fine-tuning for text classification (2018). https://doi.org/10.48550/ARXIV.1801.06146, https://arxiv.org/abs/1801.06146

[16] Jiang, X., Havaei, M., Chartrand, G., Chouaib, H., Vincent, T., Jesson, A., Chapados, N., Matwin, S.: On the importance of attention in meta-learning for few-shot text classification. arXiv preprint arXiv:1806.00852 (2018)

[17] Kanewala, U., Bieman, J.M.: Using machine learning techniques to detect metamorphic relations for programs without test oracles. In: 2013 IEEE 24th Intl. Symposium on Software Reliability Engineering (ISSRE). pp. 1–10. IEEE (2013)

[18] Kanewala, U., Bieman, J.M., Ben-Hur, A.: Predicting metamorphic relations for testing scientific software: a machine learning approach using graph kernels. Software testing, verification and reliability **26**(3), 245–269 (2016)

[19] Kaplan, J., McCandlish, S., Henighan, T., Brown, T.B., Chess, B., Child, R., Gray, S., Radford, A., Wu, J., Amodei, D.: Scaling laws for neural language models. arXiv preprint arXiv:2001.08361 (2020)

[20] Müller, S., Gogoll, V., Vu, A.D., Kehrer, T., Grunske, L.: Automatically finding metamorphic relations in computational material science parsers. In: International Workshop on Software Engineering for eScience (SE4eScience) (2022)

[21] Peng, Z., Lin, X., Niu, N., Abdul-Aziz, O.I.: I/o associations in scientific software: A study of swmm. In: Intl. Conf. on Computational Science. pp. 375–389 (2021)

[22] Peng, Z., Lin, X., Santhoshkumar, S.N., Niu, N., Kanewala, U.: Learning i/o variables from scientific software's user manuals. In: International Conference on Computational Science. pp. 503–516. Springer (2022)

[23] Radford, A., Wu, J., Child, R., Luan, D., Amodei, D., Sutskever, I., et al.: Language models are unsupervised multitask learners. OpenAI blog **1**(8), 9 (2019)

[24] Ren, X., He, W., Qu, M., Huang, L., Ji, H., Han, J.: Afet: Automatic fine-grained entity typing by hierarchical partial-label embedding. In: Proceedings of the 2016 conference on empirical methods in natural language processing. pp. 1369–1378 (2016)

[25] Reynolds, L., McDonell, K.: Prompt programming for large language models: Beyond the few-shot paradigm. In: Extended Abstracts, CHI. pp. 1–7 (2021)

[26] Rossman, L.A.: 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), epa.gov/sites/production/files/2019-02/documents/epaswmm51manualmaster8-2-15.pdf

[27] Santos, E.A., Campbell, J.C., Patel, D., Hindle, A., Amaral, J.N.: Syntax and sensibility: Using language models to detect and correct syntax errors. In: IEEE 25th Intl. Conf. on Softw. Analysis, Evolution and Reeng. pp. 311–322 (2018)

[28] Segura, S., Fraser, G., Sanchez, A.B., Ruiz-Cortés, A.: A survey on metamorphic testing. IEEE Transactions on software engineering **42**(9), 805–824 (2016)

[29] Su, F.H., Bell, J., Murphy, C., Kaiser, G.: Dynamic inference of likely metamorphic properties to support differential testing. In: 2015 IEEE/ACM 10th International Workshop on Automation of Software Test. pp. 55–59. IEEE (2015)

[30] Vu, A.D., Kehrer, T., Tsigkanos, C.: 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. pp. 182:1–182:5. ACM (2022)

[31] Wang, B.: Mesh-Transformer-JAX: Model-Parallel Implementation of Transformer Language Model with JAX. github.com/kingoflolz/mesh-transformer-jax

[32] Wang, B., Komatsuzaki, A.: GPT-J-6B: A 6 Billion Parameter Autoregressive Language Model. github.com/kingoflolz/mesh-transformer-jax (2022)

[33] Xu, F.F., Alon, U., Neubig, G., Hellendoorn, V.J.: A systematic evaluation of large language models of code. In: Proceedings of the 6th ACM SIGPLAN Intl. Symposium on Machine Programming. pp. 1–10 (2022)