Metamorphic Testing of AI-based Applications: A Critical Review

Muhammad Nadeem Khokhar[1]
Dept. of Computer Science
SZABIST
Islamabad, Pakistan

Muhammad Bilal Bashir*[2]
Computing & Technology Dept.
IQRA University
Islamabad, Pakistan

Muhammad Fiaz[3]
Computing & Technology Dept.
IQRA University
Islamabad, Pakistan

***Abstract*—Metamorphic testing is the youngest testing approach among other members of the testing family. It is designed to test software, which are complex in nature and it is difficult to compute test oracle for them against a given set of inputs. Metamorphic testing approach tests the software with the help of metamorphic relations that guide the tester to check if the observed output can be produced after applying a certain input. Since its first appearance, a lot of research has been done to check its effectiveness on different complex families of software applications like search engines, compilers, artificial intelligence (AI) and so on. Artificial intelligence has gained immense attention due to its successfully application in many of the computer science and even other domains like medical science, social science, economic, and so on. AI-based applications are quite complex in nature as compared to other conventional software applications and because of that they are hard to test. We have selected specifically testing of AI-based applications for this research study. Although all the researchers claim to propose the best set of metamorphic relations to test AI-based applications but that still needs to be verified. In this study, we have performed a critical review supported by rigorous set of parameters that we have prepared after thorough literature survey. The survey shows that researchers have applied metamorphic testing on applications that are either based on Genetic Algorithm (GA) or Machine Learning (ML). Our analysis has helped us identifying the strengths and weaknesses of the proposed approaches. Research still needs to be done to design a generalized set of metamorphic rules that can test a family of AI applications rather than just one. The findings are supported by strong arguments and justified with logical reasoning. The identified problem domains can be targeted by the researchers in future to further enhance the capabilities of metamorphic testing and its range of applications.**

Keywords*—Metamorphic testing; metamorphic relation; test oracle problem; artificial intelligence; genetic algorithm; machine learning

I. Introduction

Before using any type of machinery or equipment, it is necessary to make sure that it is needed, that it is accurate, and that its output is in line with the requirements of whatever process it will be used in. The same applies to the selection of any material or device for any given procedure. When it comes to the use of software, this process of testing and evaluating becomes even more important. Each software is in some way contributing to the performance of a system. What this means, in essence, is that testing software allows one to make sure it is fulfilling its role properly and thus allowing the system to run efficiently and without error. As such, to test a software is to test the system and to test the system is to make sure that the system will function as it should.

In the world we live in today, many of our day to day activities are being completed with the help of machines and a large amount of data and procedures have been digitized and automated, completely or partially, and this is continuing to grow. It is sometimes even believed that machine learning will continue to advance until it can attain near-human proportions and thus automate as well as improve many aspects of our lives. This in turn means that much in our everyday lives depends on the smooth and flawless functioning of these software and as such the need to test them by maintaining an input to output log is ever more important. Some software are extremely difficult to test because of their application and complex input scenarios like intelligence-based 3D games, search engines, compilers, machine learning based applications and so on. For those and similar other applications, it is very difficult to compute expected outputs for a set of given inputs. This problem is known as test oracle problem [26] and conventional testing techniques are not capable to solve it.

Conventional testing approaches like data-flow testing and control flow testing require test oracle to check and compute the correctness of a software. Test oracles are quite difficult to compute in situations when we want to test complex software as stated earlier. Metamorphic testing [27] uses metamorphic relations to test complex software that cannot be easily tested due to test oracle problem. Metamorphic relations help in verifying whether the executed test cases have produced expected output or not. Since its origin researchers have proposed different techniques to apply this approach on various kinds of software. Literature shows immense variation in the domain of its applications including Web Services [9], [10], [11], [12], Computer Graphics [13], [14], [15], Embedded Systems [16], [17], [18], [19], Simulation and Modeling [20], [21], Artificial Intelligence [1], [3], [5], [2], [4], Bioinformatics [22], [23], Compilers [24], [25] and so on. Other than the aforementioned complex types of software, some authors have even proposed using metamorphic testing to support conventional structural testing and mutation testing [28], [29].

Artificial intelligence is spreading its wings and covering almost every domain of knowledge where computer-aided applications can be used. Their application base is grown with each day passing covering not only computer science but also other domains like medical science, engineering, economics, and so on. AI-based applications are difficult to test as they are extremely complex in nature (implementing

complex algorithms or may be manipulating huge amount of data) and its hard to generate test oracles for them. Due to this very reason, metamorphic testing techniques have been designed and proposed to test them. Although some studies exist and researchers do claim their proposed techniques are good but no review study exists that specifically evaluates them on the basis on sound justifications and parameters. We have conducted a thorough survey to evaluate metamorphic testing techniques and to highlight their strengths and weaknesses on the basis of well defined benchmark. The findings will guide the researchers to improve the existing techniques to further exploit the capabilities of metamorphic testing.

Rest of the paper is organized as follows. In Section II, we present metamorphic testing concepts briefly. In Section III we present related research work that has been done in the domain of artificial intelligence. Section IV describes evaluation criteria for analysis and Section V presents analysis of the literature review. We conclude in Section VI and present future directions in Section VII.

II. Metamorphic Testing

Metamorphic testing is a specific technique or method used for testing programs and machinery. This method works by identifying and testing relations (technically called Metamorphic Relations) which are generated by executing the software multiple times. Metamorphic relations actually provide a mapping of sample inputs to expected outputs without specifically stating the actual outputs. The relations are basically set of rules that show how a given input will be transformed into one or more possible outputs by the software under test. Fig. 1 explains the metamorphic testing process with the help of a flow-chart followed by its explanation.

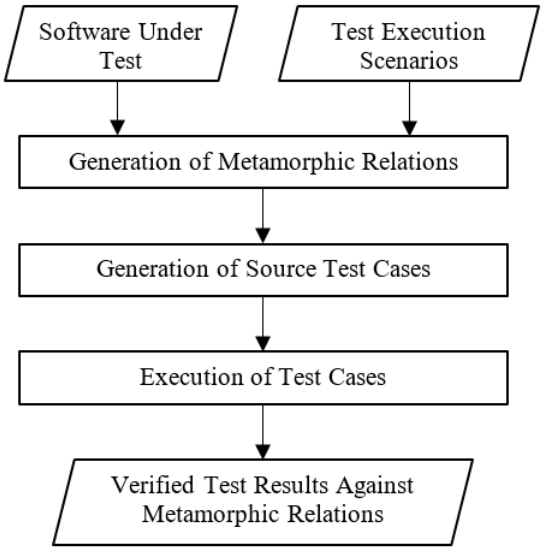

Fig. 1. Metamorphic Testing Process

The testing process begins with the input of software under test and its key usage scenarios. This helps in devising the metamorphic relations. In order to determine what Metamorphic Relations are to be used, one must first identify the key points that must be verified and validated in order for the program to work. In other words, one must identify those key points which determine the smooth functioning of the equipment, any error in which would cause the software to malfunction. As such, the metamorphic relations are found using or are built on these key points. On the basis of metamorphic relations, tester then can generate source test cases that will be run on the software to validate its correctness. Test cases are generated for every execution scenario with the help of corresponding metamorphic relation. Then they are executed and results are recorded, which later can be verified with the help of metamorphic relations to ensure if a give test case is passed or failed.

III. Related Work

In this section, we will briefly discuss various metamorphic testing methodologies that we have encountered in the literature available on testing AI-based applications. The total number of techniques and studies reviewed amounts to eight, with all of them being located within a time-frame of ten years.

Building on the basic ideas of metamorphic testing, Murphy et al. [1] have, in this paper, attempted to test the usefulness of this methods by applying it on specific algorithms and machine learning applications. They have, in their analysis, tried to identify the metamorphic properties of these applications and then to use them for the purpose of testing and for the identification of defects and faults. Based on this analysis, they have also attempted to find specific properties that could be used for the identification of metamorphic relationships and thus endow the process with much greater applicability in the area of testing.

The applications that were used for this analysis were Marti Rank and, a machine learning application that is used for the purpose of ranking, and PAYL, which is used for the purpose of detecting intrusion. A similar analysis that the authors had conducted previously was also considered, this one involving Marti Rank and SVM Light and being focused on making use of pseudo-code and runtime options. Using the observations thus obtained, the authors were able to identify the metamorphic properties of these applications and then use them for the purpose of testing. In the case of Marti Rank, for instance, the authors were able to identify and pinpoint certain problems in its code. One thing that they did, for instance, was to multiply each value in the data by -1, thus creating a new dataset that was a negative version of the original. The expected result, in this case, was that the same results would be obtained as with the original, but the cases that were considered "worst" originally would now have become the "best" and the ones that were originally taken as "best" would now have become the "worst", thus resulting in a complete inversion of the original order. Unfortunately, the developers of the application had not considered such a scenario and as such the results obtained by the authors of this study were not in accordance with the expected results. Investigating further, the authors were also able to identify the actual fault which was causing this to happen and were thus able to make use of metamorphic testing to successfully test and evaluate this application.

Similar results were obtained when testing the PAYL application, in which the authors were able to identify errors by

using data of various kinds as input. In one case, for instance, they removed all payloads of 274 bytes from the data, but the error alert that was raised by the system was not the one that was supposed to have been raised. Similarly, when they added extra payload data of 1448 bytes, the system, instead of raising a "length-never-seen-before" alert, raised both the required alert as well as an additional one that originally should not have been raised. In short, they were able to identify cases where the required outcome was not obtained as well as those in which the required outcome was accompanied by another unexpected one. As such, the authors' study was able to confirm the possibility of using metamorphic testing for the use of analyzing and evaluating Machine Learning applications.

The main goal, however, of the study was not to simply test the possibility of using metamorphic testing for the purpose of evaluating Machine Learning applications, but to also explore if any specific properties could be singled out for the general purpose of analyzing such applications. For this purpose, the authors also compared the specific properties that were used in both cases and were able to identify six specific properties that, in their words, could be found in a number of machine learning applications and could be taken as the properties which one would go for when having to test such applications. The identified metamorphic properties were multiplicative, additive, invertive, permutative, exclusive, and inclusive. The authors, on this basis, conclude that metamorphic testing may be used for the purpose of testing machine learning applications and they suggest that the six properties identified by tested and evaluated further in later studies so as to determine just how useful they might be for the purpose of metamorphic testing.

Barus et al. [2] have, in their study, attempted to deal with the "oracle problem" by making use of Metamorphic Testing. Their specific concern is with whether this testing method can be employed for the testing of heuristic methods, and for this they have applied Metamorphic Testing to a heuristic algorithm known as Greedy Algorithm. The reason why this had to be done is that heuristic algorithms are usually based on educated guesswork, experimental and use experience, and from general observations on the process. When testing such algorithms, it is not possible for one to have an exact solution, which, in turn, makes it difficult to verify the results obtained. To overcome this problem, the authors have made use of Metamorphic Testing by identifying nine separate Metamorphic Relations for the Greedy Algorithm, and then have implemented the process on five separate versions of a programme that makes use of it. In each case, they have observed that the Metamorphic Testing process was able to identify at least one fault, and, in some cases, they have also found the careful selection of appropriate Metamorphic Relations can reveal more than one failure. They also mention, at the same time, that their study has experimented with Metamorphic Testing by applying it on only one specific algorithm, and that much more needs to be done before a final conclusion may be drawn.

Machine Learning software are used oftentimes in fields relating to the computation of scientific data. These fields include, for instance, areas such as computational biology, linguistics, and so on. The problem, it would seem, is the absence of a test oracle that one might employ in the testing and validation of these software. When there is no oracle, it is impossible for one to know what the correct answer would be to a randomly provided input. As such, one is not able to test, validate, and thus rely on the use of such software, which may be involved in many crucial scientific studies. Addressing this issue, Xie et al. [3] have presented a solution to the issue of testing these Machine Learning software. They have, in their study, made use of metamorphic testing as a possible solution to the oracle problem and have also practically tested it on specific Machine Learning software. The software which they have used have been chosen on their common usage in areas relating to bioinformatics. Their essential hypothesis is that metamorphic testing procedures can be used for both verifying and validating software and algorithms. They have also delineated major problems and pitfalls that one might face when implementing metamorphic testing and have also suggested that the method they have proposed may also be used for testing and validation for software being used in fields that do not have to do with scientific computation.

The basic principle behind metamorphic testing, as has been discussed earlier, is to make use of certain properties and the relationship between the input and the output. When making a specific change to the input, the changes that will come about in the output for which can be predicted, one can detect any errors or problems when the system does not produce the expected results. In case this does happen, then there is most probably some issue in the implementation process, the software itself, or even in the selection of the algorithm. As such, the authors believe that the process can be used both to test software and to see if the software being used is the one that is required.

An alternative method is to make use of a "pseudo-oracle", which basically involves making use of various implementations of the same algorithm and observing the results the give when processing the same input. Unfortunately, this may not always be advisable, for the simple reasons that, firstly, there might be just one implementation of the algorithm, and secondly, various implementations may, regardless of whether produced by the same or by different people, may share similar faults, especially when the fault is somewhere deeper and not just in the implementation. Metamorphic testing gets around this problem by focusing instead on the various properties of the software and, on their basis, seeing if the software reacts to changes in input as it is expected to. If not, then the observer may proceed to search for and identify the problem.

The authors have, in their study, tried to gauge the benefit of using metamorphic testing in the area of Machine Learning, specifically with regards to the use of machine learning applications in areas of computational science. Their target applications are k-Nearest Neighbors (kNN) Classifier and Naïve Bayes Classifier (NBC). Both of these are used in the field of bioinformatics and are implemented in Weka. They also do not have a test oracle. Alongside their report, they have also discussed how metamorphic testing might be implemented and have pointed out common pitfalls and methods for avoiding them. They have come up with a set of metamorphic relations and techniques that could be used for the evaluation of Classifiers. They also believe that the process would be equally applicable to other machine learning software, and as such have suggested that it be used to improve the quality of the software being produced.

In what is perhaps a follow up to the previously cited study, Xie et al. [4] have attempted, once again, to evaluate metamorphic testing as a viable solution to the oracle problem. They have, in the study, discussed what metamorphic testing is, identified relations that can and cannot be of use in the testing process, delineated major faults and pitfalls, and tested the viability of the method on two specific classifiers. Alongside this, they have also conducted mutation analysis as well as cross validation. They conclude that the method proposed is quite handy when it comes to killing mutants and also that it serves as a good complement to cross-validation, which, they point out, in itself is not enough when it comes to testing classifiers.

The reason why the authors chose to work on classifiers, it would seem, is that such software are used often in scientific inquiries and are also being implemented in and becoming a part of the devices that human beings use in their everyday lives. The authors make reference to the fact that most studies try to find ways to improve machine learning processes, but the attention paid to the accurate testing and evaluation of machine learning applications has received comparatively less attention. To bridge this gap and thus pave way for further improvements and developments, they have attempted to work on precisely this issue. As far as testing the metamorphic testing process is concerned, they have tested it by using it for the evaluation and testing of two specific classifiers: k-Nearest Neighbors and Naïve Baye Classifier. In doing so, they have also tried to isolate those metamorphic relations which are most useful, or which are necessarily violated, when a problem is found. If any relation other than these "necessary" ones shows an error, it may or may not be because of an error in implementation, or, to put it more simply, it would simply represent a deviation from "expected" behavior, whereas violations in the necessary relations would always correspond to an error in implementation.

This division between necessary relations and otherwise also allows the authors to confirm whether or not metamorphic testing can be used for both verification and validation. The authors conclude, in short, that relations that are marked with "necessary properties", or those properties which help indicate the existence of faults, are useful for verification, whereas those which are not marked can be used for validation. Furthermore, the systematic nature of the proposed procedure increases its applicability overall. The method, successful as it has been in its application on the software used for the study, may just as easily be used to test other software, and one need not have a detailed understanding of the software, its problems, and testing procedures, but work on a simple understanding of the test procedure. Furthermore, the process has also been shown to be quite effective in killing mutations, which further proves its reliability, and can also be used as a good complement to cross-validation, which, in itself is not sufficient as a means for verification.

One must keep in mind, however, that metamorphic testing is used for testing software, and as such it has one specific limitation: it reveals faults, it does not prove correctness. It is possible that one received no errors but the software or the algorithm chosen are not appropriate for the specific task. Furthermore, it is also important to note that metamorphic testing relies on one's choice of appropriate relations. It is possible for one to overlook some crucial factors in this case, and as such, the process is, despite all of its benefits, only usable for the purpose of finding and identifying faults and does not help us decide whether or not the software is working perfecting or if it even is the software we should be using. Furthermore, the authors have tested the method on two specific software, and future work might be needed to test it on other software as well, and in doing so corroborate or add to these observations.

Evolutionary Testing is a process that allows one to automate and systematize the testing process and thus make it much more efficient. It works by generating test data through the use of genetic algorithms. These algorithms work by searching for best fit specific solutions or functions through a process that mirrors the biological theory of evolution. In a world where machines are playing an increasingly important role, both in research and in everyday areas of work, it is important that these software be tested properly so that they can deliver results and thus keep the system from falling apart. In their study Dong et al. [5] have attempted to improve the process of evolutionary testing by involving metamorphic relations during the construction of fitness functions. The purpose of evolutionary testing, as was mentioned earlier, is to automate and systematize the testing process. It works by making use of genetic algorithms for the purpose of generating test data. This data is then tested in accordance with a fitness function, which, in turn, is based on the specifics of the software's structure. The algorithm continues to renew the test input population until an optimum has been achieved. The authors have attempted to add to the current iteration of this process by making use of metamorphic testing principles. Since metamorphic testing allows one to overcome the oracle problem, the authors believe that considering metamorphic relations during the construction of fitness functions would allow one to improve the evolutionary testing procedure. After testing this hypothesis, they conclude that their "improved" version of the evolutionary testing process is superior in three ways: firstly, it requires a smaller amount of iterations in reaching the optimum; secondly, it has a higher hit percent; and lastly, it enlarges the search region and the test area, thus making it better when it comes to detecting faults.

Yoo [6] has, in his work, proposed a testing procedure which combines metamorphic testing with statistical approaches, thus creating a test procedure for stochastic optimization algorithms. The problem with stochastic optimization algorithms is many-fold. First of all, there base algorithms themselves come in may forms and each iteration can be different from the other. Secondly, they are non-testable, in that their results cannot be predicted and as such one cannot and does not have a test oracle to compare to. Thirdly, even metamorphic testing cannot be applied to such algorithms without facing certain challenges, namely, that the nature of these algorithms makes direct comparison problematic and that the results obtained are affected not just by implementation but also by the specific problem being considered. The first challenge is addressed primarily by combining metamorphic testing procedures with statistical hypothesis testing, creating a new variant of metamorphic testing that has been dubbed as "statistical" metamorphic testing. As for the second problem, the author proposes, in this study, to observe and evaluate the effect that different problem instances have on the results

obtained and the usefulness of the procedure. The author tests the proposed procedure by applying it to a specific algorithm called Next Release Problem. Multiple instances of this algorithm are used and tested on various datasets, some of them real and some of them artificially prepared. Faults have also been placed in the algorithm to see how the procedure responds. This has been done through a mutation tool known as Mujava. The overall conclusion of the work is that the procedure can effectively be used for the identification of specific types of faults. Some, however, were identified as non-killable mutations. Finally, the author proposes that further study be used to identify effective metamorphic relations and also to test the procedure on more complicated and sophisticated algorithms.

Another study on the testing and evaluation of genetic (and, by extension, other randomised) algorithms was conducted by Rounds et al. [7]. Their primary motive for doing so was that even though such software are used in testing and optimizing machine learning applications, very little has been done on the process of testing these algorithms themselves. This problem becomes even more serious when one takes in to consideration the fact that many of these are being used to test and evaluate software in areas of everyday life, including even such sensitive and critical areas as medicine, traffic, and so on.

The main problem in testing genetic algorithms, it would seem, is that these algorithms are "randomised", or, to put it simply, there is no fixed answer that they can be tested against. Each time the software is run, the answer is different. The very goal of these software, it would seem, is to find optimal solutions by testing and evaluating a string of possible answers. In other words, one cannot, for the purpose of evaluation, compare the final answer to a predefined "correct" one, and if the data provided is in any way misleading or problematic, then it will become even more difficult to get the desired output. The authors propose to deal with this situation by making use of metamorphic testing, a process that has already been used on some comparatively less complex applications and thus to propose a solution to the given problem. Their suggested process combines metamorphic testing with statistical testing (aka "statistical" metamorphic testing) to create a relatively more reliable process. This process is then evaluated in a number of ways, particularly through mutation testing, or the generation of "mutants" by altering a single line of code and then testing to see if the software is able to identify it or not.

Through their study, the authors were able to identify a set of metamorphic relations that, in their observation, are more useful than the others. They also checked these relations in relation to genetic algorithms, genetic algorithm operators, and algorithms involving differential evolution. They have, in total, identified 17 different relations, 5 of which are useful for fitness functions, 9 for the entire genetic algorithm, and 3 for the operators. They also compare the performance of these relations for deterministic unit tests, concluding that the metamorphic relations have a higher mutation score. They also found a specific relation that worked well with a specific function and two that had to be replaced with new ones. All in all, they consider their work to have been successful and suggest that future work test these relations with other genetic algorithms and operators and that more relations be established for the testing of various applications, algorithms, and implementations.

Dwarakanath et al. [8] have also evaluated the usefulness of the metamorphic testing process for the purpose of testing machine learning applications. The reason why they opted to do that was, like many other before them, because of the widespread and ever-increasing involvement of machine learning applications in matters of everyday life and society. The problem, however, is that while such applications are used for many different everyday purposes, they cannot be tested by comparing results to some sort of an "expected" output, the reasons primarily being that: firstly, the application would involve too many possible inputs for one to be able to create a test; secondly, that the specific output for a given input itself may not be known; and finally, because an incorrect result may not really be caused by a bug, but may also have been caused by one of three other possible reasons—insufficient training, poor architecture, and incorrect function. As such, the authors have tried to apply metamorphic testing to deal with this issue by testing it on two specific programs: support vector machines (with both linear and non-linear kernel functions) and an image classifier that makes use of deep learning, the latter of which also makes use of a convolutional neural network in its processing. The actual process of testing involved the introduction of artificially produced bugs, a technique that is more commonly known as Mutation testing, from which it was observed that around 71% were successfully identified. The researchers have also tried to develop certain metamorphic relations for the analysis and testing of these and other applications, with the ones developed for the SVM being able to identify all 12 mutants and the one designed for the ResNet application (i.e. the classifier making use of this specific convoluted neural network) was able to identify half of its 16 mutants. The authors, however, also mention that the complexity of the ResNet application forced them to stick with a subset of the actual data for the current experiment, and that future work would have to rely on a greater chunk of and maybe even the complete dataset and also that the relations defined would have to be tested for other similar applications.

IV. Evaluation Criteria

In this section we present evaluation criteria by which we analyze the surveyed literature on the metamorphic testing of various AI-based techniques. We have discussed each of the parameters from the devised benchmark in the discussion below.

A. Metamorphic Relations

Metamorphic testing aims at solving the test oracle problem. In metamorphic testing, we identify certain metamorphic relations that we can later on use to generate test cases. These test cases should conform to the identified metamorphic relations in order to show the correctness of the software that is being tested. Failure to conform with these relations may help us uncovering logical bug(s) in the software. Techniques that make use of metamorphic relations are more useful in our opinion than the ones that do not. Table I shows the evaluation criteria for this parameter.

TABLE I. EVALUATION CRITERIA FOR METAMORPHIC RELATIONS

Value	Criteria
Yes	The value Yes means that the technique has identified the metamorphic relations and are presented in the paper.
No	The value No means that the technique has not identified and presented the metamorphic relations.

B. Literature Reviewed

A research study with thorough literature review gives the reader more confidence in the findings of the study. As such, we have included this parameter to evaluate the selected research studies on the basis of the amount of literature that has been reviewed by the researchers in trying to identify the research gap. Table II gives the evaluation criteria for Literature Reviewed.

TABLE II. EVALUATION CRITERIA FOR LITERATURE REVIEWED

Value	Criteria
Low	The value Low means the researchers have reviewed and cited less than or equal to 10 research studies.
Medium	The value Medium means the researchers have reviewed and cited more than 10 and less than or equal to 30 research studies.
High	The value High means the researchers have reviewed and cited more than 30 research studies.

C. Domain of the Subject

The domain of the subject is important to consider because every domain has its own attributes and strengths that separate it from others, and which, as a result, cause a change or variation in the specific requirements of the testing process. During this study, we have found two different domains: Meta-heuristics (Genetic Algorithm) and Machine Learning. Table III describes the criteria for this parameter.

TABLE III. EVALUATION CRITERIA FOR DOMAIN OF THE SUBJECT

Value	Criteria
GA	The study has used Metamorphic Testing on Genetic Algorithms.
ML	The study has used Metamorphic Testing on Machine Learning or deep Learning.

D. Complexity

Once a new technique has been successfully tested, it is very important to note how complex it is for a new user willing to use or work on the technique and thus to learn and master it. The more complex a technique, the less the chances will be of it being adopted as compared to any other comparatively less complex technique. Table IV presents the criteria for this parameter.

E. Time Efficiency

The time taken by a technique to complete a task is one of the most important factors for evaluation. Techniques that save more time are considered to be more efficient than those that are not able to save as much. Table V gives the evaluation criteria for this parameter.

TABLE IV. EVALUATION CRITERIA FOR COMPLEXITY

Value	Criteria
Low	It is very easy to learn and master the technique. The time required can range from minutes to couple of hours.
Medium	It takes average efforts and time to learn and master the technique. The time required may range from more than couple of hours to 5-6 hours on average.
High	It is very difficult to learn and master the technique. The time required on average by a person exceeds 6 hours.

TABLE V. EVALUATION CRITERIA FOR TIME EFFICIENCY

Value	Criteria
Low	The value Low means that by applying this technique not a lot of time is saved. One of the reasons can be the complexity of the technique itself that requires a lot of effort and time.
Average	The Average value means that on some applications the technique will help in saving time and in some applications it will not.
High	The High value means that the technique is so comprehensive and quite easy to learn and offers some tool support that helps in saving a lot of testing effort.

F. Automation

An automation serves as the proof of a concept and it also helps in reducing the time and effort required in testing. In other words, a technique that offers automation is considered better than one that does not. Table VI provides evaluation criteria for this parameter.

TABLE VI. EVALUATION CRITERIA FOR AUTOMATION

Value	Criteria
Yes	The value Yes means the technique has either developed a new tool or has customized an existing one to validate the approach.
No	The value No means the technique has neither developed a new tool nor has customized any existing. Hence, no validation.

G. Case Study

The number of case studies determines how well the proposed technique has been tested. A larger number of case studies means that the technique has been tested more often and as such it provides one with a guarantee regarding the flexibility of this technique. A such, it provides reassurance that the technique is reliable and that it can be used for a varying number of situations. Table VII gives the evaluation criteria for this parameter.

TABLE VII. EVALUATION CRITERIA FOR CASE STUDY

Value	Criteria
Low	if number of case studies used in the research study for validation is less than 10, then the value Low will be assigned.
Medium	If total number of case studies used for validation range between 11 and 30, then Medium value will be assigned.
High	The value High will be assigned if total number of case studies used for validation are more than 30.

V. ANALYSIS

In this section we present our analysis on the use of metamorphic testing on AI-based applications related to both Machine Learning and Genetic Algorithm. We analyze the techniques proposed in different studies on the basis of the seven parameters discussed in the evaluation criteria section above. Table VIII gives the summary of the analysis in a tabular form for easy understanding.

Among the main limitations of metamorphic testing is the highly wide spectrum of the possible "Metamorphic Relations" that need to be identified. Even within the same domain, it is not necessary that the same Relations can be used for different applications. This makes it difficult for testers to learn how to identify and short-select the "Relations" that are beneficial in testing. None of the surveyed techniques [1], [3], [4], [7], [5], [2], [6], [8] have paid attention of devising generic set of metamorphic relations that can be applied on every of large set of AI-based applications. This seems to be one of the biggest limitations in existing research.

A Reasonable number of papers have been studied in each case, giving more reliability to the findings of the study. Although all of them have developed tools to validate the results, but most of the automated solutions are usually prototype and are not available for download and use. In that case users (testers) have to either develop them on their own or have to shift to other available tools even though they may not be as effective as they should be.

In case users (testers) opt to develop the most recent or comprehensive technique so they can use it to test AI-based applications, they find it difficult to understand them. It has been observed that the techniques used are very hard to learn except proposed by Dong et al. [5]. This high learning curve makes it very difficult for new researchers and testers to learn and implement the techniques and as such becomes problematic when it comes to wide-spread use.

Testing these days require huge amount of effort not only because of the complexity, which can be inherent but also due to the size of he software under test. So reducing the total effort required has been a concern for the researchers and testers. The time efficiency achieved is also low or medium in all of these studies, with the exception of only [3], [7]. Researchers need to pay some attention in the domain of metamorphic testing to make it more effective and practical for use.

Using a large number of case studies gives more authenticity and confidence in the technique developed. Unfortunately, in most of the studies, this number is found to be low and sometimes medium, which, in turn, raises questions on the applicability of the techniques on a wider scale.

The comforting fact, however, is that metamorphic testing seems to be effective in both Machine Learning and Genetic Algorithm domains, giving room for more research in the area and thus creating possibilities.

VI. CONCLUSION

Testing AI-based applications is a highly challenging task because these applications may not necessarily follow a certain known pattern for every set of input. Consequently, the output might differ for each execution. Most of the time, it is not possible for one to be able to determine the correct output for a particular input. Moreover, a change of input may result in an unexpected change in the output. As such, what is required is an approach that exploits the properties of these application to generate accurate and reliable transformation functions. One such approach is called metamorphic testing. This approach works by identifying some "Relations" that can be used to test applications that work on data sets having ambiguous accuracy. Although the identification of these relations is a challenge and the learning curve for new testers is high, metamorphic testing is still a very effective approach that provides one with an opportunity to overcome the issues of testing such applications. After conducting detailed survey and analysis we have found that although some research has been done in the domain of Machine Learning and Genetic Algorithm but none of them provides a standard set of generic metamorphic relations that can be applied on all or a large number of applications. Other than that metamorphic testing has not been applied on other AI techniques like Deep Learning and other optimization algorithms.

VII. FUTURE WORK

Very little has been done when it comes to the application of Metamorphic Testing on AI applications, and as such there is a lot of room for researchers to work on new techniques that can be adopted very easily by testers and which can be applied very easily on a large number of case studies.

REFERENCES

[1] C. Murphy, G. E. Kaiser, and L. Hu, "Properties of machine learning applications for use in metamorphic testing", 2008.

[2] A. Barus, T. Y. Chen, D. Grant, F.-C. Kuo, and M. F. Lau, "Testing of heuristic methods: A case study of greedy algorithm", in IFIP Central and East European Conference on Software Engineering Techniques, pp. 246-260, 2008.

[3] X. Xie, J. Ho, C. Murphy, G. Kaiser, B. Xu, and T. Y. Chen, "Application of metamorphic testing to supervised classifiers", in 2009 Ninth International Conference on Quality Software, pp. 135-144, 2009.

[4] X. Xie, J. W. Ho, C. Murphy, G. Kaiser, B. Xu, and T. Y. Chen, "Testing and validating machine learning classifiers by metamorphic testing", Journal of Systems and Software, vol. 84, pp. 544-558, 2010.

[5] G. Dong, S. Wu, G. Wang, T. Guo, and Y. Huang, "Security assurance with metamorphic testing and genetic algorithm", in 2010 IEEE/WIC/ACM International Conference on Web Intelligence and Intelligent Agent Technology, pp. 397-401, 2010.

[6] S. Yoo, "Metamorphic testing of stochastic optimisation", in 2010 Third International Conference on Software Testing, Verification, and Validation Workshops, pp. 192-201, 2010.

TABLE VIII. ANALYSIS TABLE OF METAMORPHIC TESTING OF AI-BASED APPLICATIONS

Technique	Metamorphic Relations	Literature Reviewed	Domain	Complexity	Time Efficiency	Automation	Case Study
Murphy et al. [1]	Yes	Medium	ML	High	Medium	Yes	Medium
Barus et al. [2]	Yes	Medium	GA	High	Medium	Yes	Medium
Xie et al. [3]	Yes	Medium	ML	High	High	Yes	Low
Xie et al. [4]	Yes	High	ML	High	Low	Yes	Low
Dong et al. [5]	Yes	Medium	GA	Medium	Medium	Yes	Medium
S. Yoo [6]	Yes	High	GA	High	Medium	Yes	Low
Rounds et al. [7]	Yes	High	GA	High	High	Yes	Medium
Dwarakanath et al. [8]	Yes	High	ML	High	Low	Yes	Low

[7] J. Rounds and U. Kanewala, "Systematic Testing of Genetic Algorithms: A Metamorphic Testing based Approach", arXiv preprint arXiv:1808.01033, 2018.

[8] A. Dwarakanath, M. Ahuja, S. Sikand, R. M. Rao, R. Bose, N. Dubash, and S. Podder, "Identifying implementation bugs in machine learning based image classifiers using metamorphic testing", in Proceedings of the 27th ACM SIGSOFT International Symposium on Software Testing and Analysis, pp. 118-128, 2018.

[9] W. K. Chan, S. C. Cheung, and K. R. P. Leung, "Towards a metamorphic testing methodology for service-oriented software applications," in Fifth International Conference on Quality Software, pp. 470-476, September 2005.

[10] W. K. Chan, S. C. Cheung, and K. R. P. H. Leung, "A metamorphic testing approach for online testing of service-oriented software applications." International Journal of Web Services Research, vol. 4, no. 2, pp. 61–81, 2007.

[11] C. Sun, G. Wang, B. Mu, H. Liu, Z. Wang, and T. Y. Chen, "A metamorphic relation-based approach to testing web services without oracles," International Journal of Web Services Research, vol. 9, no. 1, pp. 51–73, Jan. 2012.

[12] C. Castro-Cabrera and I. Medina-Bulo, "An approach to metamorphic testing for ws-bpel compositions," in Proceedings of the International Conference on e-Business (ICE-B), pp.1–6, July 2011.

[13] J. Mayer and R. Guderlei, "On random testing of image processing applications," in Sixth International Conference on Quality Software, pp. 85–92, October 2006.

[14] R. Guderlei and J. Mayer, "Towards automatic testing of imaging software by means of random and metamorphic testing," International Journal of Software Engineering and Knowledge Engineering, vol. 17, no. 06, pp. 757–781, 2007.

[15] W. K. Chan, J. C. F. Ho, and T. H. Tse, "Piping classification to metamorphic testing: An empirical study towards better effectiveness for the identification of failures in mesh simplification programs," in 31st Annual International Computer Software and Applications Conference, pp. 397–404, vol. 1, July 2007.

[16] T. H. Tse, S. S. Yau, W. K. Chan, H. Lu, and T. Y. Chen, "Testing context-sensitive middleware-based software applications," in Computer Software and Applications Conference, 2004. COMPSAC 2004. Proceedings of the 28th Annual International, pp. 458–466 vol.1, September 2004.

[17] W. K. Chan, T. Y. Chen, H. Lu, T. H. Tse, and S. S. Yau, "A metamorphic approach to integration testing of context-sensitive middleware-based applications," in Fifth International Conference on Quality Software, pp. 241–249, September 2005.

[18] W. K. Chan, T. Y. Chen, S. C. Cheung, T. H. Tse, and Z. Zhang, "Towards the testing of power-aware software applications for wireless sensor networks," in Ada Europe 2007 - Reliable Software Technologies, ser. Lecture Notes in Computer Science, N. Abdennadher and F. Kordon, Eds. Springer Berlin Heidelberg, vol. 4498, pp. 84–99, 2007.

[19] F.-C. Kuo, T. Y. Chen, and W. K. Tam, "Testing embedded software by metamorphic testing: A wireless metering system case study," in IEEE 36th Conference on Local Computer Networks (LCN), pp. 291–294, October 2011.

[20] K. Y. Sim, W. K. S. Pao, and C. Lin, "Metamorphic testing using geometric interrogation technique and its application," in Proceedings of the 2nd International Conference of Electrical Engineering/Electronics, Computer, Telecommunications, and Information Technology, pp. 91–95, 2005.

[21] T. Y. Chen, F.-C. Kuo, H. Liu, and S. Wang, "Conformance testing of network simulators based on metamorphic testing technique," in Formal Techniques for Distributed Systems, ser. Lecture Notes in Computer Science, D. Lee, A. Lopes, and A. Poetzsch-Heffter, Eds. Springer Berlin Heidelberg, vol. 5522, pp. 243–248, 2009.

[22] L. L. Pullum and O. Ozmen, "Early results from metamorphic testing of epidemiological models," in ASE/IEEE International Conference on BioMedical Computing (BioMedCom), pp. 62–67, December 2012.

[23] A. Ramanathan, C. A. Steed, and L. L. Pullum, "Verification of compartmental epidemiological models using metamorphic testing, model checking and visual analytics," in ASE/IEEE International Conference on BioMedical Computing (BioMedCom), pp. 68–73, December 2012.

[24] Q. Tao, W. Wu, C. Zhao, and W. Shen, "An automatic testing approach for compiler based on metamorphic testing technique," in 17th Asia Pacific Software Engineering Conference (APSEC), pp. 270–279, November 2010.

[25] V. Le, M. Afshari, and Z. Su, "Compiler validation via equivalence modulo inputs," in Proceedings of the 35th ACM SIGPLAN Conference on Programming Language Design and Implementation, ser. PLDI '14. New York, NY, USA: ACM, pp. 216–226, 2014.

[26] E. T. Barr, M. Harman, P. McMinn, M. Shahbaz, and S. Yoo, "The oracle problem in software testing: A survey," Software Engineering, IEEE Transactions on, vol. 41, no. 5, pp. 507–525, May 2015.

[27] T. Y. Chen, S. C. Cheung, and S. M. Yiu, "Metamorphic testing: a new approach for generating next test cases", Technical Report HKUST-CS98-01, Department of Computer Science, Hong Kong University of Science and Technology, Hong Kong, 1998.

[28] G. Dong, S. Wu, G. Wang, T. Guo, and Y. Huang, "Security assurance with metamorphic testing and genetic algorithm," in IEEE/WIC/ACM International Conference onWeb Intelligence and Intelligent Agent Technology (WI-IAT), pp. 397–401, vol. 3, August 2010.

[29] T. Y. Chen, T. H. Tse, and Z. Q. Zhou, "Fault-based testing without the need of oracles," Information & Software Technology, vol. 45, no. 1, pp. 1–9, 2003.