Using Fourier Analysis and Mutant Clustering to Accelerate DNN Mutation Testing

Ali Ghanbari
Auburn University
Auburn, AL, USA
ghanbari@auburn.edu

Sasan Tavakkol
Google Research
Irvine, CA, USA
tavakkol@google.com

***Abstract*—Deep neural network (DNN) mutation analysis is a promising approach to evaluating test set adequacy. Due to the large number of generated mutants that must be tested on large datasets, mutation analysis is costly. In this paper, we present a technique, named DM#, for accelerating DNN mutation testing using Fourier analysis. The key insight is that DNN outputs are real-valued functions suitable for Fourier analysis that can be leveraged to quantify mutant behavior using only a few data points. DM# uses the quantified mutant behavior to cluster the mutants so that the ones with similar behavior fall into the same group. A representative from each group is then selected for testing, and the result of the test, *e.g.*, whether the mutant is killed or survived, is reused for all other mutants represented by the selected mutant, obviating the need for testing other mutants. 14 DNN models of sizes ranging from thousands to millions of parameters, trained on different datasets, are used to evaluate DM# and compare it to several baseline techniques. Our results provide empirical evidence on the effectiveness of DM# in accelerating mutation testing by 28.38%, on average, at the average cost of only 0.72% error in mutation score. Moreover, on average, DM# incurs 11.78, 15.16, and 114.36 times less mutation score error compared to random mutant selection, boundary sample selection, and random sample selection techniques, respectively, while generally offering comparable speed-up.**

***Index Terms*—DNN, FFT, Mutation, Testing, Acceleration**

I. Introduction

Deep learning [1], enabled by *deep neural networks* (DNNs), has been used in modern software systems in many domains. The growing applications of DNNs in safety and mission critical systems, such as autonomous driving, healthcare, and energy, and the need for accuracy and robustness of such systems, warrant research on quality assurance of DNNs. Among many methods for quality assurance of DNNs [2], [3], testing is a widely used approach [4], wherein test data points are manually curated, or automatically generated, to satisfy certain test requirements. However, good performance on the test dataset does not necessarily imply the robustness and generality of a DNN model, and a systematic way for assessing the quality of the test data is needed.

In recent years, *mutation analysis* [5], [6] has been reintroduced in the context of DNNs [7], [8], [9], [10] as a promising method for assessing the quality of test data (see §II-A more details). Despite its promise for test data quality assurance, DNN mutation analysis remains prohibitively expensive due to its need for testing many mutants on large datasets, limiting its practical deployment [8], [11], [12], [13], [14], [15], [16]. In the realm of conventional programs, *i.e.*, programs that are not strictly based on data-driven pipelines, there is a large body of work concerning acceleration of mutation analysis [17], [18]. However, since mutation analysis is relatively new for DNNs, this important topic, which could benefit many useful techniques relying on mutation analysis, has not received as much attention.

We present a novel method, named DM#, for accelerating DNN mutation analysis by *testing fewer mutants*, which can be applied to both model-level and source-level mutation analysis. This is achieved through efficiently clustering the mutants into sets of behaviorally similar mutants and testing representatives of each cluster to approximate the mutation score. Our key insight is that DNN outputs, being real-valued and continuous, lend themselves naturally to Fourier analysis [19], [20]. This allows us to characterize the behavior of each mutant with only a handful of test samples. Specifically, we use Fourier analysis to compute a comparable signature of the overall behavior of the mutants using a *tiny fraction* of the test dataset. This way, DM# reasons about the behavior of the mutants, *e.g.*, whether they are likely killed or survived, solely based on their Fourier analysis results, obviating the need for the more expensive process of applying the mutants on the entire test dataset.

Given a set of mutants to be tested, DM# uses a tiny subset of the test dataset (*e.g.*, 0.1% of the data points) to calculate discrete Fourier transform spectra *via* an efficient algorithm, known as fast Fourier transform [21] (FFT). FFT spectra uniquely encode the behavior of functions as vectors of frequency bins and their amplitudes. DM# uses the similarity between FFT spectra as a metric for quantifying behavioral similarity of the corresponding mutants, which is then used to cluster the mutants into sets with likely identical mutation results, *e.g.*, all killed or all survived. Finally, a representative mutant from each cluster is selected for full testing, *i.e.*, testing with all the test data points. The mutation testing result of the representative is then reused for all the mutants it represents.

We have implemented DM# [22], which is applicable to a wide range of supervised classifier DNN architectures, such as fully-connected neural networks (FCNNs), convolutional neural networks (CNNs) with/without residual blocks, and recurrent neural networks (RNNs). DM# complements the existing work [11], [12], [13], [14], [15], [16] and can be used alongside them for further acceleration. To increase its usability

2643-1572/25/$31.00 ©2025 IEEE
DOI 10.1109/ASE63991.2025.00175

as a tool, we have designed DM# not to require any parameters from the user to operate, instead it leverages lightweight search procedures for finding appropriate parameter values for FFT analysis, as well as clustering (see §IV for more details).

We empirically study DM# using a set of 14 DNN models of varying sizes, ranging from thousands to millions of parameters, representative of a broad range of models with FCNN, CNN, and RNN architectures. Our experiment with DM# consists of four parts. First, we use a small subset of our benchmark models, consisting of simple FCNN models, to provide empirical evidence that DM# components are well-behaved. As a byproduct of this phase, we also obtain heuristic default values for guiding the parameter search procedures such that DM# incurs a maximum of 5% error in mutation score and achieves a minimum of 10% speed-up.

Next, we apply DM#, with its default parameters, to another subset of benchmark models containing larger and more complex DNN models, representative of real-world DNN models. The evaluation results provide empirical evidence on the effectiveness of DM# in reducing mutation testing time without incurring significant error in mutation score. Specifically, DM# reduces the number of mutants to be tested by 35.71%, on average, which translates into 28.38% reduction in end-to-end mutation testing time (when the overhead of DM# itself is also taken into account), while incurring only an average of 0.72% error in mutation score. Furthermore, we compare DM# to three baselines techniques: random mutant selection [13] (RMS), boundary sample selection [15] (BSS), and random sample selection (RSS). We observed that while RMS, BSS, and RSS are 1.28, 2.38, 2.91 times, respectively, faster than DM#, DM# incurs 11.78, 15.16, and 114.36 times less mutation score error than RMS, BSS, and RSS, respectively.

Lastly, we examine whether FFT-based clustering in DM# captures meaningful behavioral similarities rather than reducing to mutant output histograms, and whether propagating test outcomes from cluster representatives to other mutants remains reliable. Skipping FFT analysis increases mutation score error by over $24\times$. We also confirm that clustering results can be reliably propagated from representatives to other mutants (see §V for details).

In summary, this paper makes the following contributions.

- **Concept:** We introduce the concept of applying FFT to quantify the behavior of DNN mutants with few data. While we have explored clustering of the mutants induced by FFT spectra for accelerating mutation testing, many other applications, *e.g.*, model compression and inference optimization, are also conceivable.
- **Implementation:** We have implemented DM# [22], which can be used as a mutation testing add-on for the existing frameworks, such as DeepMutation [8] or DeepCrime [9], with a bit of integration coding.
- **Empirical evaluation**: We present an empirical evaluation of DM# on a diverse set of DNN models, and a comparison to the related work. DM# reduces mutation testing time with negligible error in mutation score. It also outperforms baseline techniques in terms of mutation score accuracy, while offering comparable speed-up.

II. Background

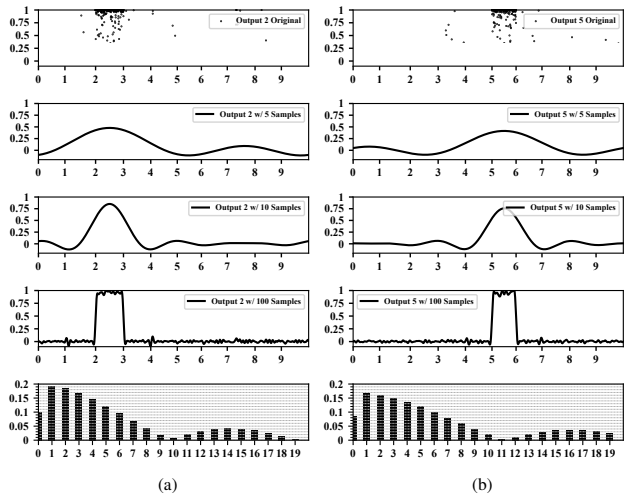
A. Mutation Analysis

Mutation analysis [5], [6] is a program analysis method for assessing test suite quality in conventional programs. Central to this method is the set of program transformation operators known as *mutation operators*, or *mutators*. Mutation analysis involves generating a set of program variants, called *mutants*, by systematically mutating program elements using mutators, and running the test suite on the mutants to check if the outputs of the mutants are different from that of the original program; if different, the mutant is marked as *killed*, otherwise as *survived*. Test suite quality is measured by *mutation score*, which is traditionally approximated by the ratio of killed mutants over survived mutants. Higher mutation scores indicate stronger suites. While mutation analysis, or its acceleration, does not directly find bugs, it helps strengthening testing by measuring and improving test suite quality. Beyond its original application in assessing the quality of test suites, mutation analysis has had many other applications [23], [24].

Recently mutation analysis has been applied in the context of DNNs [7], [8], [9], [10]. Like its conventional counterpart, since its introduction as a test data quality assessment method, DNN mutation analysis has had plenty of applications, including adversarial sample detection [25] and generation [26], robustness analysis [27], [28], aiding manual labeling of test data *via* prioritization of test data [29], accuracy estimation to alleviate the need for manual labeling of test data [30], fault localization [13], automated repair of DNNs [31], [32], modular decomposition of DNN models [16], and improving the quality of test dataset by generating new data points guided by mutation testing [33], [34], [35].

DNN mutation analysis is usually done at two different levels: (1) source code and the data used to specify and train a model, known as *source-level mutation analysis*; (2) the graph representing the trained model itself, known as *model-level mutation analysis*. Both forms of mutation analysis are costly [27], [36], so there is a recent research trend in accelerating this process [11], [12], [13], [14], [15], [16], [37]. The two forms differ from each other mainly in the way the mutants are generated, the former involves mutating the source and/or the training data and training the resulting mutants, while the latter directly mutates an already trained model. The testing of the mutants in both forms are identical to each other, so an approach for reducing the costs of mutation testing is readily applicable in both contexts.

To study the impact of the presented acceleration technique on the mutation score, we use the mutation score formula by Ma *et al.* [8]: $\frac{1}{|M|\times|L|}\sum_{\mu\in M}|\text{killingLabels}(\mu)|$, where M is the set of mutants that is obtained by applying a set of mutators on a given DNN model m. $L = \{\text{label(t)} \mid t \in T\}$ is the set of labels in a test dataset T, wherein the function 'label' returns the ground-truth label for the data points in T. For any mutant $\mu \in M$, killingLabels(μ) is defined to be



(a) (b)

Fig. 1: Row 1: outputs 2 and 5 of a DNN classifier trained on MNIST dataset in columns (a) and (b), respectively. Rows 2-4: Fourier approximation of the two functions using 5, 10, and 100 terms, respectively. Last row: bar charts for the first 20 frequency buckets of the FFT for the two functions. The bar charts are annotated with gray guidelines to aid visual comparison of the heights of the bars between two diagrams.

$\{\text{label}(t) \mid t \in T \text{ and } \text{kill}(\mu, t)\}$. A mutant μ is said to be *killed* by a test data point $t \in T$, denoted by the predicate $\text{kill}(\mu, t)$, if $\text{argmax}(m(t)) = \text{label}(t)$ and $\text{argmax}(\mu(t)) \neq \text{label}(t)$, where $\text{argmax}(m(t))$, or $\text{argmax}(\mu(t))$, denotes the label predicted by the model m, or its mutant μ, for t.

B. Fourier Analysis

Fourier series represent periodic, piece-wise continuous functions using a series consisting of a constant term and infinitely many sine and cosine terms. The sine and cosine terms are harmonically related, *i.e.*, their frequencies are integer multiples, or harmonics, of the so-called fundamental frequency, which is calculated based on the period of the original function. The *Fourier transform* extends this frequency-domain representation to piece-wise continuous functions that are not periodic. In short, Fourier transform transforms a given piece-wise continuous function $f(\bar{x})$ into a complex-valued continuous function $\hat{f}(\omega)$ over frequencies. This new function represents the amplitudes and phases of the different frequency components ω that make up the original function $f(\bar{x})$. *Discrete Fourier transform* is the discrete counterpart of Fourier transform that is used to calculate discrete frequency components based on a set of values sampled from the original function $f(\bar{x})$. This notion further extends frequency-domain representation to discrete and non-continuous functions. This paper uses *fast Fourier transform* (FFT), which is an efficient (and famously elegant) implementation of discrete Fourier transform. The output of the FFT is often called *spectra*, as it defines the magnitude of each frequency bucket. The original function can be uniquely reconstructed from the FFT spectra through a process known as *inverse FFT*. Readers are referred to [20], [19], [21] for more information about Fourier analysis.

FFT has extensively been used in understanding waveforms, solving differential equations, image compression, audio processing, and even DNN compression and optimization [38], [39], [40]. This paper, for the first time, proposes the use of FFT in accelerating mutation analysis of DNNs. A key motivation behind using FFT, rather than other methods [41], [42], [43], is that FFT converges very quickly and enables representing a holistic view of the behavior of a complex function with only a handful of samples.

III. Motivating Example

The main idea behind DM# is that FFT spectra can be used to efficiently quantify the behavior of complex real-valued functions such as DNNs. This enables comparing different DNNs, *e.g.*, different mutants of a DNN, to identify the ones that behave similarly, *e.g.*, they are likely to survive or be killed by certain inputs. To understand this idea better, let us look into a simplified example. Assume that we have trained an FCNN model with four layers, each with 50 neurons with ReLU activation function. After training the model using MNIST dataset, it achieves a test accuracy of 0.9727. This model has 10 softmax outputs; the first row in Fig. 1 plots the third and sixth outputs of the model (corresponding to the output/class indices 2 and 5, respectively) in a plane where the x-axis comprises 10,000 MNIST test images arranged and sorted in ascending order of their labels, and the y-axis ranges between 0 and 1, representing softmax values.

Assume further that we take FFT of these two outputs using 5, 10, and 100 randomly selected samples that include at least one data point of the class corresponding to the plotted output. The FFT spectra obtained *via* 5, 10, and 100 samples can be used to reconstruct the original functions. Rows 2 to 4 in Fig. 1 plot the reconstructed functions using 5, 10, and 100 samples, respectively. This is a visual demonstration of how well the original function is approximated using only 0.05%, 0.1%, and 1% of the images in the test dataset. In fact, had we performed the same process for all outputs of the model, the reconstructed model using 5, 10, and 100 samples would achieve test accuracies of 0.8953, 0.9877, and 0.9931, respectively, which are quite close to the original model.

Our goal here is not to reconstruct the original functions, rather we want DM# to directly use the calculated spectra to compare different mutants of a DNN. This is possible thanks to the fact that FFT spectra uniquely characterize functions and similar functions are expected to have similar amplitudes in each of the matching frequency buckets. The last row in Fig. 1 depicts the bar charts for the first 20 frequency buckets of the FFT spectra for the outputs 2 and 5 of the model which are obtained using 100 samples. In these diagrams, the position of each bar represents a frequency bucket, while the height of the bars represents the amplitude for that frequency. Given that using only 100 samples the two functions are perceived as phase-shifted "pulse functions," the FFT spectra of the two functions are rather similar to each other.

DM# views FFT spectra of the mutants of a DNN model as points in a multi-dimensional space, and leverages the distance

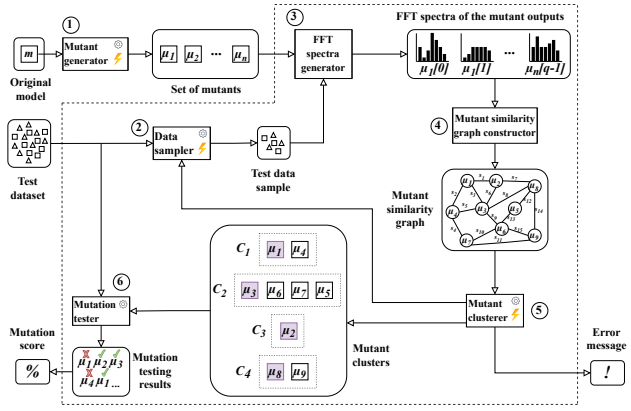


Fig. 2: A DNN mutation analysis workflow involving DM#. Sharp-edged rectangles denote processes; rounded ones denote data/artifacts. Arrows denote control flow. Gear and lightning bolt icons are used to annotate user-configurable and non-deterministic processes, resp. Processes and artifacts inside the area marked by dashed line are part of DM#.

between the points to calculate similarity values that are then used to cluster the mutants into groups of mutants with likely the same outcome, *e.g.*, all survived or all killed.

IV. DM# Approach

DM# reduces mutation testing costs by clustering mutants based on their approximated behavior and testing a representative from each cluster instead of all mutants. A DNN classifier and its mutants comprise functions with common inputs, *i.e.*, one function per output. For example, a 10-class classifier DNN model, or any of its mutants, is a collection of 10 functions (one output per class) with the same input. The key insight of this paper is that DNN outputs are real-valued functions suitable for FFT analysis, enabling the computation of comparable signatures for mutant behavior. DM# clusters mutants using the magnitude of FFTs, grouping those with similar behavior. Specifically, it employs a similarity measure based on the maximum Euclidean distance between the FFT magnitudes of mutant outputs. This method, commonly used in digital signal processing [44], [45], must be applied carefully due to differences between sound signals and mutant outputs. Mutants arise from small variations targeting specific neurons in a DNN. Since individual neurons contribute to specific outputs, meaning mutations affect only a subset of outputs while others remain nearly unchanged. Consequently, their FFTs share many harmonics, leading to zero, or extremely small, Euclidean distances. To measure mutant distances, DM# takes the maximum Euclidean distance of the outputs, emphasizing the most impacted output. Formally, given two mutants μ_1 and μ_2 with q outputs and a test set S, the distance is defined to be $\delta_S(\mu_1, \mu_2) = \max_{0 \le i < q} \left\{ \sqrt{\sum_{j=0}^{|S|} (\|FFT_S(\mu_1[i])\|[j] - \|FFT_S(\mu_2[i])\|[j])^2} \right\}$, where FFT_S computes the FFT of the i-th output using S, $[.]$ denotes vector/array indexing, and $\|.\|$ represents FFT magnitude. The Euclidean distance between FFT magnitudes is computed for each output, and their maximum is used as the mutant distance.

The sample set S is randomly drawn from a *tiny* subset of the test dataset, *e.g.*, 0.1% of data points. For a meaningful approximation of mutant behavior, the sample should ideally include at least one instance per class. For example, in a 10-class dataset like MNIST with 10,000 test points, each class has hundreds of instances. A 50-point sample (*i.e.*, 0.5% of the dataset) should thus contain 5 points per class. Analyzing DM# in pathological cases where certain classes are missing is left for future work.

DM# leverages FFT-based distances to cluster mutants with likely identical outcomes, *e.g.*, all surviving or getting killed. However, most clustering algorithms require a similarity measure rather than raw distances. To address this, we map distance values to similarity values *via* an exponentiation function, and define the similarity of mutants μ_1 and μ_2 as:

$$\sigma_S(\mu_1, \mu_2) = e^{-\delta_S(\mu_1, \mu_2)} \tag{1}$$

Intuitively, for a given sample set S, $\sigma_S(\mu_1, \mu_2) = 1$, if the mutants μ_1, μ_2 are identical in behavior, and the value of the σ_S function quickly approaches to zero as the distance between the mutants behavior grows. This rate of shrinking in exponentiation function helps spreading similarity values between 0 and 1, better than other alternatives, *e.g.*, reciprocal, which helps interpretability of the similarity values.

With this definition of mutant similarity, we now examine DNN mutation analysis workflows incorporating DM# as an accelerator. Fig. 2 illustrates such a workflow. Mutants generated by a host mutation analysis system, along with the test dataset, are passed to DM#, which returns results indicating which mutants are killed or survive. The host system then utilizes these results for tasks like computing mutation scores or fault localization. The following sections detail each process and artifact in the order they appear in the workflow.

A. Mutation Generator

The mutation generator (① in Fig. 2) is a component of DNN mutation analysis systems like DeepMutation [8] or DeepCrime [9]. It generates mutants by applying mutators to the original model or its defining code/data. For model-level mutation, mutators directly produce mutants, while source-level mutation requires training mutated code/data to generate executable mutants. While this paper focuses on model-level mutation, DM# is applicable to source-level analysis as well.

The mutation generator is external to DM#, providing one of its inputs. In this paper, we employ DeepMutation's model-level mutation to generate mutants from trained models, and we use a subset of mutators, namely Gaussian Fuzzing, Weight Shuffle, Neuron Effect Block, Neuron Activation Inverse, and Neuron Switch. These mutators, unlike DeepMutation's layer-level mutators, produce higher quality, non-trivial mutants [36] that are used in later works [27], [25], [16]. Mutators are parameterized and involve randomness. In this paper, we use DeepMutation's default parameter values.

B. Data Sampler

The data sampler (② in Fig. 2) is a DM# component that selects a small subset of the test dataset for downstream FFT analysis (see §IV-C). Typically, only a tiny fraction, *e.g.*, 0.1%, of the original test dataset, which may contain thousands of data points, is sampled. Sampling is crucial for acceleration in DM#, as it enables estimating mutant behavior without applying them to the entire test dataset.

We design DM#'s data sampler to randomly select at least one instance from each class, ensuring that the FFT spectra accurately represent the mutant outputs. The selection of an instance from a given class is currently done randomly, but various selection strategies deserve a deeper empirical analysis in future extensions of this work. A parameter, x, called the *data sampling rate*, determines how many instances per class should be selected. We observed that the in-class variations of the sampled data points due to randomness, say which image of 5 in the MNIST dataset should be selected, rarely impact the outcome of the mutation analysis. However, if too few samples are selected and a mutant misclassifies only those specific instances, the resulting FFT spectra may not accurately reflect its behavior. Conversely, a mutant could correctly classify just the sampled instances, making the spectra appear as if from a perfect classifier. To mitigate this randomness, we repeat all our experiments five times.

By default, DM# does not require the user to specify the x value. Instead, it iteratively invokes this component along with the mutant clusterer (see §IV-E) using different x values from the set $\{1, 3, 5, 10, 20, 30, 40, 50, 100, 200, 300\}$, selecting 1, 3, 5, *etc*., samples per class—until the user-specified mutant reduction goal is met. However, users can disable this search loop and manually specify x if they have better insights into the appropriate number of samples per class for their specific application. For instance, in RQ1 in §V-D1, we disabled the loop to supply x values from a predefined range and study the behavior of DM# at each value.

C. FFT Spectra Generator

Given a set of mutants M and a set of test samples S, the FFT spectra generator (③ in Fig. 2) produces FFT spectra for each output of every mutant in M. These spectra are then used in the next step (see §IV-D). For illustration, the FFT spectra are represented as histograms labeled $\mu_1[0], \mu_1[1], \ldots, \mu_n[q-1]$ in the figure. Specifically, the histogram labeled $\mu_j[i]$ represents the FFT spectrum of output index i of the mutant $\mu_j \in M$, computed using the test dataset sample S. The FFT process takes a series of output values (representing a discrete function) and a set of samples as input, producing a $|S|$-dimensional vector where each element represents the magnitude of a frequency bucket. The functions provided to this component correspond to the individual outputs of mutants in M, while the sample set S is drawn from the test dataset. Applying FFT on output index i of mutant μ_j using S is denoted as $FFT_S(\mu_j[i])$ in the definition of δ_S.

A naïve application of FFT on mutant outputs would be costly, because given n mutants, each with q outputs, the FFT process with S samples would need to be repeated nq times. Each round would involve applying each mutant to the data points in S, resulting in $|S|nq$ mutant applications, which can be expensive for mutants with large outputs. DM# circumvents this cost by applying each mutant to each data point only once and reusing the outputs in subsequent iterations. Specifically, DM# applies n mutants to the data points in S, stores the results, and uses the memoized outputs when a specific mutant's output i is needed during the FFT calculation. This optimization reduces the cost of this step by a factor of q.

D. Mutant Similarity Graph Constructor

Mutant similarity graph constructor (④ in Fig. 2) leverages the FFT spectra for the mutants outputs to construct a *mutant similarity graph*, defined below.

Definition 1 (Mutant Similarity Graph): Mutant similarity graph is a complete weighted undirected simple graph with vertices M and edges E. M contains all and only the generated mutants, and for each $\mu_1, \mu_2 \in M$, with $\mu_1 \neq \mu_2$, there is an edge $(\{\mu_1, \mu_2\}, w) \in E$, where w, called the weight, is defined to be $\sigma_S(\mu_1, \mu_2)$ for a test dataset sample set S. ■

The mutant similarity graph is complete, *i.e.*, it contains an edge between every distinct pair of mutants. Since σ_S is symmetric (*i.e.*, $\sigma_S(\mu_1, \mu_2) = \sigma_S(\mu_2, \mu_1)$ for any non-empty S), there is a unique edge between each distinct pair, making the graph undirected and simple. The mutant similarity graph constructor builds this graph by iterating over mutant pairs, calculating their similarities using the σ_S function, and adding edges to an efficient C++-based graph data structure. The σ_S function relies on the δ_S function, which is defined using the FFT_S function. The values of FFT_S are pre-calculated for each mutant output, making graph construction fast, despite the time complexity of $O(|M|^2 q|S|)$, where q is the number of mutant outputs.

E. Mutant Clusterer

The mutant clusterer (⑤ in Fig. 2) clusters nodes in the mutant similarity graph into similarity groups. This component of DM# uses parallel hierarchical agglomerative clustering (ParHAC) [46], a scalable and efficient version of the traditional hierarchical agglomerative clustering algorithm. Scalability is important for DM#'s performance, as mutant similarity graphs are typically huge.

Clustering in ParHAC is guided by the *linkage threshold* parameter, denoted τ. Selecting appropriate τ value is crucial for the DM#'s performance: a well-chosen τ can lead to significant time savings with minimal impact on mutation score, while a poor choice can make DM# slower than the vanilla approach for some models. However, finding the optimal τ can be challenging, reducing the tool's usability. We, therefore, automated finding τ value by implementing a binary search over the entire search space of valid linkage threshold values, *i.e.*, the interval (0, 1). The algorithm seeks a τ value that satisfies a user-defined constraint, called the *mutant reduction constraint*, which specifies a desired reduction in mutant percentage, defined by the range $[l, h]$, where $0 \leq l \leq h \leq 1$. Since

Algorithm 1 Parameter search procedure in DM#

Input: Number of mutants N, mutant reduction goal $R = [l, h]$
Output: Set C of clusters satisfying the mutant reduction constraint R, or an error message if R is not achievable

```
1: for x ∈ {1, 3, 5, 10, 20, 30, 40, 50, 100, 200, 300} do
2:     τ_lo ← 0
3:     τ_hi ← 1
4:     while 10^−5 ≤ (τ_lo + (τ_hi − τ_lo)/2) ≤ 0.99999 do
5:         τ ← τ_lo + (τ_hi − τ_lo)/2
6:         C ← DoParHACClustering(G_x, τ)
7:         mut_red_rate ← (N − |C|)/N
8:         if mut_red_rate < l then
9:             τ_hi ← τ
10:        else if mut_red_rate > h then
11:            τ_lo ← τ
12:        else
13:            return C
14:        end if
15:    end while
16: end for
17: return "Mutant reduction goal not satisfiable"
```

one representative from each cluster is selected for testing (see §IV-F), the reduction in mutants equals the number of clusters. Thus, mutant reduction is calculated as $\frac{|M|-|C|}{|M|}$, where $|M|$ is the total number of mutants and $|C|$ is the number of mutant clusters.

Selecting the right τ value is closely tied to choosing the correct data sampling rate x, as the sampling rate affects how well FFT spectra approximate mutants' behavior. For this, DM# performs a linear search over the set $\{1, 3, 5, 10, 20, 30, 40, 50, 100, 200, 300\}$ to select 1, 3, 5, *etc*., samples from each class and then conducts a binary search for τ values for each chosen x. Values 1, 3, and 5 represent *small* sample sizes, 10, 20, 30, 30, 40, and 50 represent *mid-size* sample sizes, while 100, 200, and 300 represent *large* sample sizes. Algorithm 1 formalizes the steps DM# takes to coordinate the data sampler, FFT spectra generator, mutant similarity graph constructor, and mutant clusterer. It takes the number of mutants N (*i.e.*, $|M|$) and the mutant reduction constraint R as inputs, returning the clusters that satisfy R or an error if the search fails. The for-loop at Line 1 tries different x values. For each x, DM# performs data sampling, FFT analysis, and generates a mutant similarity graph, denoted G_x. In lines 2-15, a binary search for τ values is performed to satisfy R. The variables τ_{lo} and τ_{hi} represent the lower and upper bounds for τ. The while-loop at Line 4 checks if the midpoint falls within the practical range (0, 1). Values below 10^{-5} are treated as zero, and values above 0.99999 as one, making the clustering algorithm to produce a single cluster or singleton clusters, respectively. This defines the binary search termination condition, which, on average, terminates in 18.4 iterations, as per our experiments. Lines 5-7 assign the midpoint to τ, run the ParHAC clustering with τ, and calculate the mutant reduction rate. In Lines 8-14, if too many clusters are created, the upper bound is decreased; if too few, the lower bound is increased. If a valid set of clusters is found, it is returned. If all x values are exhausted and no suitable τ is found, an error message is returned at Line 17.

Monotonicity of the mutant reduction rate with respect to the linkage threshold is essential for Algorithm 1 to function properly. While we do not prove this formally, our empirical analysis in RQ1 strongly supports this claim. Additionally, this analysis led to selecting the range [0.26, 0.56] as a heuristic default for the mutant reduction constraint, which is expected to reduce mutation testing time by at least 10% with no more than 5% error in mutation score. This allows DM# users to avoid specifying any parameters.

Mutant clusterer outputs a list of clusters with their representatives. To create this list, it starts with an empty list, iterates over the clusters from Algorithm 1, randomly selects a representative node from each cluster, and adds the representative-cluster pair to the list. Other selection strategies, such as choosing the node with the highest degree, are left for future work. To account for randomness in representative selection and the ParHAC algorithm, we repeated the experiments five times.

F. Mutation Tester

The mutation tester (⑥ in Fig. 2) produces the output of DM#. It tests the cluster representatives identified by the Mutant Clusterer using the full, unsampled test dataset provided to DM#. The output is a list of pairs (μ, ψ), where μ is a mutant and ψ represents its status. Depending on DM# configuration, ψ can be a Boolean indicating whether μ is killed or survived, or an integer representing the size of the killingLabels set for μ as defined in §II-A. DM# calculates ψ for each representative and replicates it for all mutants in that cluster. This approach, adapted from past clustering-based mutation analysis acceleration techniques [47], [48], [49], marks all mutants in the cluster as killed if the representative is killed, and survived if the representative survives. Mutation tester is user-configurable, allowing users to define a Python function to calculate a mutant's status, either as killed/survived or by returning the size of the killingLabels set. For the experiments in this paper, the output is a list of pairs $(\mu, |\text{killingLabels}(\mu)|)$, which is used in the mutation score calculation as defined in §II-A.

V. Experiments

We investigate the following research questions (RQs).

- **RQ1 (Monotonicity and Heuristic Parameter Values):**
  1) How does the mutant reduction rate vary with the linkage threshold (τ) across different data sampling rates (x), and is this relationship monotonic?
  2) What range of mutant reduction rates results in a maximum of 5% error in mutation score and a minimum of 10% reduction in mutation testing time?
- **RQ2 (Effectiveness):**
  1) How does DM#, with default configuration, perform compared to the vanilla approach on larger models?
  2) How does DM#, with default configuration, compare to other baseline approaches?
- **RQ3 (Soundness and Predictive Accuracy):**
  1) Does DM# provide meaningful behavioral clustering that goes beyond simple analysis of output classes over the selected test inputs?

Table 1: Benchmark for the experiments. Train # and Test # represent the number of data points in the train and test datasets, respectively, and Cls # denotes the number of classes/labels in the test dataset.

Architecture	Dataset	Scope	Size	Train #	Test #	Cls #	Test Acc.
FCNN	EMNIST	RQ1	45,676	124,800	20,800	26	0.8831
	FMNIST		44,860	60,000	10,000	10	0.8755
	KMNIST		44,860	60,000	10,000	10	0.8704
	MNIST		44,860	60,000	10,000	10	0.9754
LeNet-5	EMNIST	RQ2-3	45,786	124,800	20,800	26	0.9228
	SVHN		62,006	73,257	26,032	10	0.8537
ResNet-10	Caltech-101		5,033,446	7,316	1,828	102	0.721
	CIFAR-10		4,986,250	50,000	10,000	10	0.8236
MobileNetV2	Caltech-101		334,886	7,316	1,828	102	0.6351
	CIFAR-10		287,690	50,000	10,000	10	0.7185
EfficientNetB2	CIFAR-100		7,915,101	50,000	10,000	100	0.8175
	Dogs		7,943,281	12,000	8,580	120	0.8119
RNN	IMDB		2,642,562	25,000	25,000	2	0.812
	Reuters		2,648,282	8,982	2,246	90	0.642

2) How accurate are the propagated test results from cluster representatives to other mutants?

The ParHAC algorithm used in DM# takes a linkage threshold parameter, τ, which is tuned *via* the binary search algorithm in §IV. In the first part of RQ1, we provide empirical evidence supporting monotonicity of mutant reduction rate with respect to τ, which is essential for the binary search algorithm to function correctly. Analyzing 1,045 data points from four models, *we found strong empirical evidence of a consistent monotonic relationship between the mutant reduction rate and τ, with no significant fluctuations*. Given this evidence, we expect that the aforementioned binary search to find τ value satisfying a mutant reduction constraint to terminate.

In the second part of RQ1, we use our measurement to determine heuristic default values for the mutant reduction rate, aiming to guide the binary search toward a rate that ensures no more than 5% error in mutation score, yet at least a 10% reduction in mutation testing time–thresholds deemed acceptable by prior work on mutation analysis acceleration [11], [12], [14], [15]. Analyzing 1,045 data points from four models, *we found that a mutant reduction constraint of* $[0.26, 0.56]$ *achieves over 10% reduction with less than 5% mutation score loss*. We adopt this as the default heuristic for RQ2-3.

In RQ2, we first study the performance of DM# on more realistic DNN models. The results provide empirical evidence on the effectiveness of DM# in *accelerating mutation testing by 28.38%, on average, at the average cost of only 0.72% error in mutation score*. Next we compare DM# to three baseline approaches, namely random mutant selection [13], boundary sample selection [15], and random sample selection. We observed that, on average, *DM# incurs 11.78, 15.16, and 114.36 times less mutation score error compared to random mutant selection, boundary sample selection technique, and random sample selection, respectively, while offering comparable speed-up in most of the cases*.

In RQ3, we study the effectiveness of two major components of DM#: FFT spectra generation and mutant clusterer. We first address the concern that whether DM# reduces to an analysis of the histograms of the output classes over the selected set of test inputs. By disabling FFT spectra generation component, and directly clustering the mutants based on their outputs, *we observed that mutation score error rises* $24.27\times$*, confirming that FFT analysis is a key for precision in DM#*. Next, we evaluate predictive performance of mutant clustering and *found that DM# performs quite well in terms of several metrics*.

A. Benchmark and Setup

Table 1 lists the DNN models used in the experiments, where each row represents a model. We have used six types of DNN architectures in our experiments: 4-layer FCNN, LetNet-5 [50], ResNet-10 [51], MobileNetV2 [52], EfficientNetB2 [53], and RNN with LSTM layers. We have use standard LetNet-5 architecture that represents the family of CNN models with no residual blocks, such as AlexNet [54] and VGGNet [55]. We have also implemented ResNet-10, MobileNetV2, and EfficientNetB2 based on the standard architectures, representing models with residual block of different complexity and layouts. Our RNN architecture consists of an embedding layer, two LSTM layers, and an output layer with softmax activation.

We have trained these model architecture on various datasets of different complexities, such as MNIST [56], a dataset of hand-written digits with classes 0-9, Fashion MNIST [57] (FMNIST), a dataset of fashion items with 0-9, the digit section of Kuzushiji MNIST [58] (KMNIST), a dataset of hand-written Japanese digits 0-9, and SVHN [59], a dataset of real-world images for 10-class classification of digits. We trained ResNet-10 and MobileNetV2 model architectures on more complex datasets such as CIFAR-10 [60], consisting of 32×32 color images in 10 different classes, and Caltech-101 [61], comprised of 224×224 color images belonging to 102 different object categories. Meanwhile, we trained EfficientNetB2 on even more complex datasets CIFAR-100, consisting of 32×32 color images in 100 different classes, and Stanford Dogs Dataset [62], a 120-class subset of ImageNet [63]. Lastly, we used Reuters [64] and IMDB [65] datasets for training the RNN models. Reuters is a dataset for 90-class classifiers for documents with news articles, and IMDB is a dataset for binary sentiment classification for movie reviews.

We have used simpler FCCN models in RQ1, making it feasible to run DM# and vanilla mutation analysis thousands of times, while other more complex models are used in RQ2-3, as indicated in the column "Scope" in the table. In the rest of the paper, we use the combination of model name and training dataset identifier to uniquely identify each of the 12 models, *e.g.*, FCNN-FMNIST, denotes the model with FCNN architecture trained on FMNIST dataset.

We have used two identical Dell Precision workstations with AMD Ryzen Threadripper @ 2.7 GHz CPU, 1 TB of RAM, and two NVIDIA RTX A6000 GPUs to conduct our experiments. Both machines run Ubuntu 22.04.4 LTS.

B. Measures

In vanilla mutation testing, *i.e.*, exhaustive testing of generated mutants, *mutation testing time* refers to the total time required for testing the generated mutants. For DM#, this includes additional overhead from data sampling, FFT analysis, clustering, and parameter search. Mutation testing time is measured in seconds but can vary by setup, so we accompany time measurements by the *number of tested mutants* as well.

Speed-up, *i.e.*, the amount of acceleration, measures the percentage decrease in mutation testing time when using DM#, or any other mutation analysis acceleration technique, instead of the vanilla approach. It is calculated as $\frac{T_V - T_0}{T_V}$, where T_V and T_0 are the mutation testing times for the vanilla approach and mutation analysis acceleration technique, *e.g.*, DM# or BSS, respectively. A related measure, *mutant reduction*, quantifies the reduction in tested mutants. The vanilla approach always has 0 mutant reduction, as it tests all generated mutants, whereas DM# tests only one representative per cluster. Mutant reduction is calculated by $\frac{|M| - N_0}{|M|}$, where $|M|$ is the number of generated mutants and N_0 is the number tested mutants.

Another measurement conducted in this paper is the *mutation score* that is calculated using the formula in §II-A. *Mutation score error* or *loss*, *i.e.*, the percentage of deviation of accelerated mutation score from vanilla mutation score, is calculated as $\frac{|MS_V - MS_0|}{MS_V}$, where MS_V is the mutation score obtained using vanilla approach, while MS_0 represents the mutation score obtained using an acceleration technique.

Lastly, we evaluate the predictive performance of the mutant clustering algorithm *via* mean absolute error (MAE), relative MAE (RMAE), precision, recall, F1, and Matthews correlation coefficient (MCC). MAE and RMAE are computed from the actual *vs.* predicted size of the *killingLabels* set in the mutation score formula of §II-A, while the other metrics follow their standard definitions based on the confusion matrix, constructed as follows. A mutant predicted as killed and actually killed (resp., survived) is a true (resp., false) positive. A mutant predicted as survived and actually survived (resp., killed) is a true (resp., false) negative. A mutant is considered *survived* if its output matches that of the original model on all test data points; otherwise, it is considered *killed* [8].

C. Baseline Approaches

Random mutant selection [13], mutation operator selection [11], [12], test data selection [15], higher-order mutation [14], [66], neuron clustering [16], and mutant clustering [37] are various approaches in the literature for accelerating mutation analysis. Among them, random mutant selection (RMS), test data selection, and neuron clustering are more relevant to DM#, as they aim to reduce the cost of testing generated mutants rather than limiting their number. Since no working implementations of these techniques were available for our setting, we reimplemented them. We implemented RMS and boundary sample selection [15] (BSS). Since Ghanbari *et al.* [13] studied random mutant selection in mutation-based fault localization, no direct comparison of our implementation was possible, but we successfully reproduced the experiments from the original BSS paper [15]. This strengthened our confidence in the accuracy of our implementation. Lyons and Ghanbari [37] have studied mutant clustering approach for single-point mutants, we observed that when applying this method on mutants impacting more than one neuron, the method becomes slower than vanilla approach due to the increased dimensionality of points to be clustered in first-order *vs.* higher-order mutation. Therefore, we excluded this approach from our study.

We further compare DM# to two other baseline approaches: (1) clustering without FFT; (2) random sample selection (RSS). The first approach performs data sampling like DM#, but it carries out clustering directly based on mutant outputs rather than their FFTs. In the rest of this paper, we use DM#$_\star$ to denote this no-FFT variant of DM#. RSS selects subset of test data points randomly while testing all the mutants; it selects the same number of data points that DM# does for FFT analysis.

D. Results

1) Answering RQ1: To address both parts of RQ1, we applied DM# to four FCNN models trained on EMNIST, FMNIST, KMNIST, and MNIST while varying the data sampling rate, x, and linkage threshold, τ. Specifically, we ranged x over $\{1, 3, 5, 10, 20, 30, 40, 50, 100, 200, 300\}$, selecting the corresponding number of samples per class, and τ over $\{0.05, 0.1, 0.15, \ldots, 0.95\}$ to systematically analyze mutant reduction, mutation score error, and timing behavior. The first part of this RQ aims to empirically show that, across different x values, the mutant reduction rate remains monotonic or constant with respect to τ. A stable reduction rate is crucial for the binary search algorithm in DM# to return correct results. To evaluate this, we recorded the mutant reduction rate for all (x, τ) combinations, repeating the process five times to account for randomness in sampling and ParHAC clustering. This resulted in 1,045 (=$11 \times 19 \times 5$) measurements per model.

Fig. 3 plots mutation reduction rate values for different τ values, when $x = 1$, for four FCNN models. Data points from 5 runs are highlighted with 5 different colors to depict the spread of mutation reduction rate for each τ value in the runs. We can visually confirm that the mutant reduction rate is either monotonically decreasing or constant as τ increases: we observe no reversals or significant outliers across any of the runs. The same is true for plots for all other x values, which can be accessed from our replication package [22]. To verify this visual analysis, for each x, *i.e.*, for every 95 data points per model, we computed Spearman's rank correlation coefficient [67], ρ, to assess the monotonicity of the relationship between τ and mutant reduction rate. As a non-parametric measure, ρ does not assume normality or linearity: $\rho = -1$ indicates a perfectly decreasing monotonic relationship, $\rho = 1$ indicates a perfectly increasing one, and $\rho = 0$ suggests no monotonic trend (*e.g.*, fluctuation). ρ is undefined if the relationship is constant.

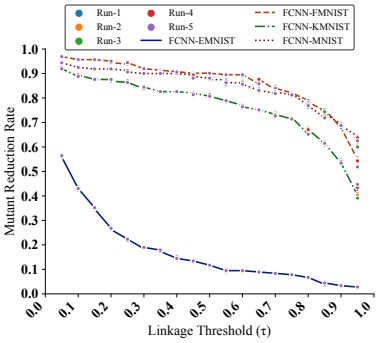


Fig. 3: Mutant reduction rate *vs.* linkage threshold when $x = 1$

Table 2 reports ρ values for different FCNN models. The magnitude of Spearman's ρ is commonly interpreted as

Table 2: Spearman's ρ values for different FCNN models obtained for different sampling rates, *i.e.*, x, and linkage threshold, *i.e.*, τ, values across 5 runs

	Sampling rate (samples per class)										
	1	**3**	**5**	**10**	**20**	**30**	**40**	**50**	**100**	**200**	**300**
FCNN-EMNIST	-0.999	-0.86	-0.832	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A
FCNN-KMNIST	-0.998	-0.999	-0.995	-0.823	N/A	N/A	N/A	N/A	N/A	N/A	N/A
FCNN-FMNIST	-0.997	-0.999	-0.998	-0.99	-0.982	-0.971	-0.855	-0.848	N/A	N/A	N/A
FCNN-MNIST	-0.994	-0.998	-0.998	-0.999	-0.999	-0.995	-0.989	-0.966	-0.958	N/A	N/A

follows [68]: $|\rho| = 1$ indicates a *perfect* monotonic relationship, $0.7 \leq |\rho| < 1$ a *strong* one, $0.4 \leq |\rho| < 0.7$ a *moderate* one, $0.2 \leq |\rho| < 0.4$ a *weak* one, and $|\rho| < 0.2$ a *very weak* or *non-monotonic* relationship. As shown in the table, all calculated ρ magnitudes exceed 0.7, mostly approaching 1. The negative sign confirms that mutant reduction rate decreases monotonically with τ, as higher linkage thresholds make clustering stricter, leading to more small clusters. Undefined ρ values (denoted "N/A") indicate a constant mutant reduction rate with respect to τ. In both cases, the results provide strong empirical evidence that mutant reduction rate remains stable as τ varies. This supports the correctness of DM# 's binary search algorithm for selecting τ under a mutant reduction constraint. Another observation we make is that as the data sampling rate increases, the relationship between mutant reduction rate and τ becomes constant for more models, as seen in the growing number of "N/A" cells from top to bottom in the table. This occurs because higher sampling rates improve FFT's approximation of mutant behavior, revealing more differences and making clustering difficult regardless of τ. In practice, a plateau in mutant reduction rate is undesirable, as it may render binary search for a suitable τ ineffective. To mitigate this, DM# searches from lower to higher sampling rates, aiming to explore diverse search spaces before performing binary search in a space that may collapse to a single value.

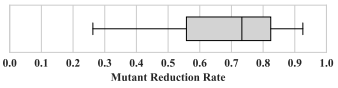


Fig. 4: Box-plot visualizing the mutant reduction rates that result in no more than 5% mutation score error and at least 10% mutation testing speed gain

An engineering goal in DM#'s design is to minimize dependence on user-defined parameters, such as x and τ, by determining them automatically. Finding the right mutant reduction constraint to balance mutation score error and speed-up is challenging and could impact usability. To address this, we analyzed measurement data from this RQ: for each model and 1,045 x and τ combinations across 5 runs, we recorded mutation score error, mutation testing time (including data sampling and clustering), and mutant reduction rate. The box plot in Fig. 4 highlights 124 of 4,180 values (1,045 per model for 4 models) that yield no more than 5% mutation score loss and at least 10% speed-up.

The minimum, 25$^{\text{th}}$ percentile, median, 75$^{\text{th}}$ percentile, and maximum mutant reduction rates are 0.26, 0.56, 0.73, 0.82, and 0.93, respectively. When setting the default mutant reduction goal, any value between the minimum and maximum could be chosen. However, just because DM# reduces approximately 82% of mutants for one model does not mean that it will achieve the same for another without greater mutation score loss. Notably, a mutant reduction constraint of [0.26, 0.56] yields over 10% reduction with less than 5% mutation score loss for more models than a higher constraint, *e.g.*, [0.56, 0.73]. Likewise, [0.56, 0.73] would work for more models than a range containing larger values, *e.g.*, [0.73, 0.93]. Thus, we conservatively select [0.26, 0.56], spanning the minimum to the 25$^{\text{th}}$ percentile. As we will see in RQ2, this constraint generalizes to more complex models, maintaining similar trends in mutation score loss and speed gain (see §V-D2). Prior mutation analysis acceleration [11], [12], [14], [15] treats $\leq$5% error as acceptable, because mutation score is itself a proxy for test adequacy and small deviations rarely affect downstream adequacy decisions. Errors below 1%, as consistently observed with DM#, can thus be considered negligible in practice.

A second view of this data wherein we plot plot mutation score error *vs.* mutant reduction rate is accessible from [22]. This corroborates that the majority of data points corresponding to $\leq$5% mutation score error fall in the range [0.26, 0.56]. To further validate these findings, we conducted another Spearman's analysis between the mutation score error and MAE/RMAE. We found strong, statistically significant positive correlations, confirming that global deviations in mutation scores mirror fine-grained deviations at the mutant level. Because mutation score error is strongly, positively correlated with both MAE and RMAE, the region [0.26, 0.56] can also be expected to exhibit the low MAE/RMAE, while achieving at least 10% speed-up. See [22] for more details.

Table 3: DM# (with default configuration) results on larger models

Model Under Test		**Mutation Score Error**			**Mutants Reduced**			**Total Time Reduction**		
Architecture	**Dataset**	**Avg**	**Min**	**Max**	**Avg**	**Min**	**Max**	**Avg**	**Min**	**Max**
LeNet-5	**EMNIST**	0.17%	0.07%	0.29%	37.23%	36.76%	37.55%	35.58%	32.96%	37.40%
	SVHN	0.71%	0.32%	1.05%	49.28%	47.68%	51.48%	29.34%	27.07%	31.35%
ResNet-10	**Caltech-101**	0.81%	0.29%	1.39%	26.74%	25.04%	29.41%	20.03%	15.22%	23.36%
	CIFAR-10	0.08%	0.01%	0.18%	42.47%	41.34%	44.06%	38.29%	35.36%	41.11%
MobileNetV2	**Caltech-101**	0.40%	0.18%	0.55%	28.07%	27.94%	28.19%	12.05%	10.07%	13.94%
	CIFAR-10	0.00%	0.00%	0.00%	36.38%	36.33%	36.41%	22.87%	20.69%	26.56%
EfficientNetB2	**CIFAR-100**	0.60%	0.08%	1.18%	27.33%	25.65%	29.96%	20.52%	15.75%	23.83%
	Dogs	0.87%	0.35%	1.45%	26.32%	24.65%	28.93%	23.76%	19.17%	26.93%
RNN	**IMDB**	1.93%	0.00%	3.53%	47.26%	39.03%	52.82%	47.13%	35.65%	49.20%
	Reuters	1.65%	0.90%	3.00%	35.97%	28.93%	54.05%	34.25%	27.39%	51.23%

2) Answering RQ2: To evaluate DM#'s effectiveness, with its default configuration, on several larger models, we enabled automatic linkage threshold search of the tool. Table 3 presents our results. The first two columns list model architecture and dataset, while subsequent columns report: (1) "Mutation Score Error" (average, minimum, and maximum in percent), (2) "Reduced Mutants" (average, minimum, and maximum in percent), and (3) "Total Time Reduction" (average, minimum, and maximum, including clustering, linkage threshold search, and sampling rate search). Results summarize five runs per model. Overall, DM# reduces the number of tested mutants by 35.71%, on average, which translates to 28.38% average reduction in mutation testing time, at the cost of only 0.72% error in average mutation score. DM# achieves this by sampling only one data point per class for all but one, *i.e.*, RNN-IMDB, model. For this model, the linear sampling search ends up selecting 10 samples per class in order to satisfy the default mutant reduction constraint [0.26, 0.56]. This is because IDBM has only two classes, and with few data points, *i.e.*, 1, 3, or 5 samples per class the mutant reduction rate goes above 56%,

prompting DM# to increase data sampling rate to increase its accuracy so that it can better distinguish mutants from one another and avoid clustering less non-equivalent mutants into one group. Despite this, DM# is able to reduce the total mutation testing time (which includes the time needed to try smaller sampling rates) by at least 35.65%.

The average number of binary search tries for finding the linkage threshold has been 18.38 (min: 1, max: 104). The outlier number of tries, *i.e.*, 104, belongs to RNN-IMDB, which requires three rounds of full binary search before finding the right linkage threshold at a sampling rate of 10 samples per class. In addition, given that the number of FFT frequency buckets used for clustering is equal to the number of test data points selected to compute the spectra, and the number of classes reported in Table 1, the number of frequency buckets used in our experiments ranged from 10 (for 10-class classifiers with $x = 1$) to 120 (for the 120-class classifier with $x = 1$).

Table 4: Comparison of DM# to the baseline approaches

Model Under Test		Average Mutation Score Error				Average Time Reduction			
Architecture	Dataset	RMS	RSS	BSS	DM#	RMS	RSS	BSS	DM#
LeNet-5	EMNIST	3.77%	96.48%	7.20%	0.17%	32.69%	69.31%	60.54%	35.58%
	SVHN	4.08%	97.98%	1.46%	0.71%	30.35%	73.65%	58.72%	29.34%
ResNet-10	Caltech-101	3.35%	84.76%	10.83%	0.81%	26.97%	73.81%	73.51%	20.03%
	CIFAR-10	2.20%	99.63%	0.33%	0.08%	27.91%	64.68%	68.05%	38.29%
MobileNetV2	Caltech-101	3.45%	93.32%	11.45%	0.40%	22.09%	25.33%	50.07%	12.05%
	CIFAR-10	3.29%	88%	0%	0%	31.81%	46.80%	39.70%	22.87%
EfficientNetB2	Dogs	3.94%	53.26%	9.68%	0.60%	30.18%	52.38%	79.10%	20.52%
	CIFAR-100	1.36%	65.43%	11.50%	0.87%	22.92%	56.52%	70.48%	23.76%
RNN	IMDB	32.42%	63.65%	32.47%	1.93%	68.57%	97.42%	96.01%	47.13%
	Reuters	27.67%	83.16%	24.46%	1.65%	69.20%	76.44%	78.36%	34.25%

Table 4 reports average mutation score error and average time reduction for four approaches RMS, RSS, BSS, and DM# (with its default configuration). For RMS, we selected 75% of the generated mutants, as the previous work [13] reports less performance drop for this setting. Similarly, for BSS, we selected the default threshold value of 10, which is reported to perform best according to the past work [15]. Meanwhile, for RSS, we selected one sample from each class, except for RNN-IMDB, where we sampled 10 data points from each class. Each tool has been executed five times and the table reports average values. The readers are referred to the replication package [22] for more details about the values, as well as RMS performance for other selection percentages.

As we can see, in this dataset, RSS consistently yields highest speed-up (63.64%, on average), as it significantly reduces the size of the test data, which is also why we get the highest loss (82.57%, on average). BSS sits in the second place, reducing total mutation testing time by 67.45%, on average. But this gain is at the cost of average 10.94% loss. Surprisingly, RMS resulted in a smaller loss of 8.50%, on average. This observation calls for further study on the power of RMS approach. We can also see that DM# consistently outperforms other techniques in terms of average loss (averaging 0.72%), and its average speed-up is comparable to that of RMS (*i.e.*, 36.27%).

Lastly, we would like to emphasize that different techniques offer distinct strengths and weaknesses, leading to a speed *vs.* accuracy tradeoff. When speed is prioritized over accuracy, RMS or BSS are well-suited, particularly for tasks such as test data augmentation with iterative methods like genetic algorithms. In such cases, the mutation score from RMS or BSS can serve as a fitness function to guide dataset improvement across iterations, after which test suite quality can be evaluated more accurately using methods like DM# or vanilla.

3) Answering RQ3: For the first part of this RQ, we disabled DM#'s FFT spectra generation component to do clustering directly based on mutant outputs. Table 5 reports our results in terms of average loss in mutation score, mutant reduction, and speed-up. While DM#$_\star$ achieves higher reduction rates

Table 5: Comparing DM# to its no-FFT variant, DM#$_\star$

Model Under Test		Avg. Loss		Avg. Mut. Red.		Avg. Speed-up	
Architecture	Dataset	DM#	DM#$_\star$	DM#	DM#$_\star$	DM#	DM#$_\star$
LeNet-5	EMNIST	0.17%	6.09%	37.23%	30.83%	35.58%	32.15%
	SVHN	0.71%	10.58%	49.28%	27.00%	29.34%	19.90%
ResNet-10	Caltech-101	0.81%	7.94%	26.74%	33.62%	20.03%	29.41%
	CIFAR-10	0.08%	5.26%	42.47%	34.52%	38.29%	33.13%
MobileNetV2	Caltech-101	0.40%	6.49%	28.07%	48.69%	12.05%	22.69%
	CIFAR-10	0.00%	5.80%	36.38%	36.37%	22.87%	21.23%
EfficientNetB2	Dogs	0.87%	26.19%	26.32%	34.55%	23.76%	33.80%
	CIFAR-100	0.60%	32.71%	27.33%	34.70%	20.52%	31.55%
RNN	IMDB	1.93%	43.29%	47.26%	46.60%	47.13%	45.98%
	Reuters	1.65%	30.90%	35.97%	35.04%	34.25%	33.65%

(*e.g.*, MobileNetV2-Caltech-101, 48.69% vs. 28.07%), this gain comes at the expense of drastically larger mutation score error, often an order of magnitude larger than DM#. For instance, on EfficientNetB2-CIFAR-100, DM#$_\star$ incurs 32.71% error compared to only 0.60% with DM#. This pattern seems to generalize: DM# maintains sub-1% error across most models, whereas DM#$_\star$ always exceeds 5%. The $24.27\times$ average difference demonstrates that FFT analysis is *not* a cosmetic step but the main driver of precision. In practice, this means that simply clustering raw mutant outputs is too coarse to preserve behavioral fidelity, especially on high-class datasets.

Table 6: Predictive performance of DM#'s mutant clusterer component

Model Under Test		MAE	RMAE	Confusion Matrix				Prec.	Rec.	F1	MCC
Architecture	Dataset			TP	FP	TN	FN				
LeNet-5	EMNIST	0.731	0.038	253	0	0	0	1	1	1	N/A
	SVHN	0.536	0.107	237	0	0	0	1	1	1	N/A
ResNet-10	Caltech-101	1.452	0.060	3111	0	0	0	1	1	1	N/A
	CIFAR-10	0.114	0.013	3021	0	0	0	1	1	1	N/A
MobileNetV2	Caltech-101	1.308	0.048	2824	0	0	0	1	1	1	N/A
	CIFAR-10	0.000	0.000	2733	0	0	0	1	1	1	N/A
EfficientNetB2	Dogs	1.418	0.059	6049	0	0	0	1	1	1	1
	CIFAR-100	1.384	0.051	6035	0	0	0	1	1	1	1
RNN	IMDB	0.175	0.175	225	19	227	44	0.922	0.836	0.877	0.760
	Reuters	0.033	0.033	140	3	459	3	0.979	0.979	0.979	0.973

DM# propagates mutation testing results of a representative to all other mutants that can introduce imprecision. To quantify this, we measure MAE and RMAE for predicted *vs.* actual killingLabels set sizes (see §II-A), and we also computed precision, recall, F1, and MCC when treating mutation outcomes in the classical kill/survive sense (see V-B). Table 6 presents these results for one run, wherein we use "N/A" to show division by zero when calculating MCC. When MAE and RMAE are considered, we observed that DM#'s clustering algorithm consistently performs well across all benchmark models: for CIFAR-10 models, the clusterer achieves near-perfect accuracy, with virtually zero error on MobileNetV2. Even on more complex datasets, such as Caltech-101 and CIFAR-100, the relative error remains below 6%, which is an evidence for generalization to more complex datasets. On LeNet-5 and RNN models, MAE and RMAE likewise remain low, which is an indication of consistent predictive performance.

When considering precision, recall, F1, and MCC, we observed that most of the mutants are killed based on classic standard, which was expected [8], [15]. In RNN models, though, we have killed and survived mutants. In these cases, the clusterer performs quite well with a precision above 90% and quite high recall, F1, and MCC. Together, these results support that DM# not only preserves mutation score at the mutation score error level but also provides reliable fine-grained predictions of individual mutant outcomes.

E. Threats to Validity

DNN mutation analysis is subject to many random factors [36], [69], *e.g.*, in the case of DeepMutation, the randomness in training and mutation generation might impact the mutation score. In this paper, our goal is not to study the reliability of mutation score as a measure for test quality, rather we aim to speed up a single run of mutation testing, so we have assumed that a fixed trained model and a set of mutants generated based off of that model are given. However, as discussed in §IV DM# itself has non-deterministic components. To account for this randomness, we have repeated our experiments five times and reported averaged results.

The set of DNN models studied in this paper is not a complete representative of the models across all applications. Given the limited resources and time, a set of models with varying sizes and structures, that the we believe are representative of the models studied in the DNN mutation analysis literature [8], [27], [9], [36], [25] are selected. For example, the benchmark contains models representing CNN models with/without residual blocks, and it also contains RNN models. DM# is open sourced, so the research community can apply it on different models and study its effectiveness on a broader range of models and mutation analysis applications.

We have obtained default values for the mutant reduction range, as well as the search space for data sampling rate, based on our observation of a limited number of models. Although these default values work satisfactorily for 10 large and complex models studies in this paper, there is no guarantee that they will generalize to other models. Alternatively, there might be different values performing better in terms of mutation score error or mutant reduction rate. DM# receives user-defined values for mutant reduction constraint and data sampling rate, so the user can try different values in case it fails.

VI. Related Work

Motivated by the many potential practical applications of DNN mutation analysis, researcher have proposed several methods for reducing the costs of this process. Some of these methods are more or less directly ported from mutation analysis of conventional programs into DNNs, while others take advantage of the unique characteristics of DNNs. For example, Feng *et al.* [11] and Wang *et al.* [12] identify sufficient subsets of existing mutators from the literature [8], [7] to avoid redundant mutants. Ghanbari *et al.* [13] adopts random mutant selection, and the technique presented by Li *et al.* [14], [66] is based on the idea of reducing the cost of mutation testing with higher-order mutants. Meanwhile, the technique introduced by Shen *et al.* [15] relies on the assumption that mutants of a DNN model are more likely to produce different results for the test data points around the decision boundary of the model, so it samples the test data that lie at the decision boundary of the model under test. Recently, Ghanbari [16] proposed to accelerate mutation analysis by generating fewer mutants through clustering the neurons. This approach together with mutant clustering based on the similarity of mutated weights have also been studied recently [37].

Klabunde *et al.* [43] survey broader methods for measuring DNN functional and representational similarities, which could also be applied to mutant clustering, but it is unclear if these methods could operate with a very few sample data points, *e.g.*, only 0.1% of the size of the test dataset, as FFT-based similarity method does. We leave investigating these alternatives in such cases for future work.

VII. Conclusions

We propose a novel technique and tool, named DM#, for accelerating DNN mutation analysis. The idea behind DM# is that DNN outputs, being real-valued functions, are amenable to FFT analysis that can be used to calculate a comparable signature of the overall behavior of the mutants using only a few test data samples. DM# uses this to cluster and mutant and test a representative from each cluster instead of testing all of the mutants. 14 DNN models, of varying sizes, trained on datasets with various complexities, have been used to study DM# from different perspectives. The results suggest that DM# reduces the number of mutants to be tested by 35.71%, on average, which translates into 28.38% reduction in end-to-end mutation testing time, while incurring only an average of 0.72% mutation score error. We further compared DM# to three baselines techniques, RMS, BSS, and RSS. We observed that while RMS, BSS, and RSS are 1.28, 2.38, and 2.91 times, respectively, faster than DM#, DM# incurs 11.78, 15.16, and 114.36 times less mutation score error than RMS, BSS, and RSS, respectively. Lastly, we demonstrated that DM# does not collapse to a trivial histogram-based analysis, and we confirmed that the clustering results can be propagated with high predictive accuracy.

Data Availability

Our measurements, as well as implementations of DM#, RMS, BSS, and RSS, are available in a replication package [22].

Acknowledgments

We would like to thank Anonymous ASE 2025 Reviewers for their valuable feedback. The first author is partially supported by the NSF grant #2446393. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the NSF or their employers.

REFERENCES

[1] Y. LeCun, Y. Bengio, and G. E. Hinton, "Deep learning," *Nat.*, vol. 521, no. 7553, pp. 436–444, 2015. [Online]. Available: https://doi.org/10.1038/nature14539

[2] C. Liu, T. Arnon, C. Lazarus, C. A. Strong, C. W. Barrett, and M. J. Kochenderfer, "Algorithms for verifying deep neural networks," *Found. Trends Optim.*, vol. 4, no. 3-4, pp. 244–404, 2021. [Online]. Available: https://doi.org/10.1561/2400000035

[3] X. Huang, D. Kroening, W. Ruan, J. Sharp, Y. Sun, E. Thamo, M. Wu, and X. Yi, "A survey of safety and trustworthiness of deep neural networks: Verification, testing, adversarial attack and defence, and interpretability," *Comput. Sci. Rev.*, vol. 37, p. 100270, 2020. [Online]. Available: https://doi.org/10.1016/j.cosrev.2020.100270

[4] J. M. Zhang, M. Harman, L. Ma, and Y. Liu, "Machine learning testing: Survey, landscapes and horizons," *IEEE Trans. Software Eng.*, vol. 48, no. 2, pp. 1–36, 2022. [Online]. Available: https://doi.org/10.1109/TSE.2019.2962027

[5] R. A. DeMillo, R. J. Lipton, and F. G. Sayward, "Hints on test data selection: Help for the practicing programmer," *Computer*, vol. 11, no. 4, pp. 34–41, 1978. [Online]. Available: https://doi.org/10.1109/C-M.1978.218136

[6] R. G. Hamlet, "Testing programs with the aid of a compiler," *IEEE Trans. Software Eng.*, vol. 3, no. 4, pp. 279–290, 1977. [Online]. Available: https://doi.org/10.1109/TSE.1977.231145

[7] W. Shen, J. Wan, and Z. Chen, "Munn: Mutation analysis of neural networks," in *2018 IEEE International Conference on Software Quality, Reliability and Security Companion, QRS Companion 2018, Lisbon, Portugal, July 16-20, 2018*. IEEE, 2018, pp. 108–115. [Online]. Available: https://doi.org/10.1109/QRS-C.2018.00032

[8] L. Ma, F. Zhang, J. Sun, M. Xue, B. Li, F. Juefei-Xu, C. Xie, L. Li, Y. Liu, J. Zhao, and Y. Wang, "Deepmutation: Mutation testing of deep learning systems," in *29th IEEE International Symposium on Software Reliability Engineering, ISSRE 2018, Memphis, TN, USA, October 15-18, 2018*, S. Ghosh, R. Natella, B. Cukic, R. S. Poston, and N. Laranjeiro, Eds. IEEE Computer Society, 2018, pp. 100–111. [Online]. Available: https://doi.org/10.1109/ISSRE.2018.00021

[9] N. Humbatova, G. Jahangirova, and P. Tonella, "Deepcrime: mutation testing of deep learning systems based on real faults," in *ISSTA '21: 30th ACM SIGSOFT International Symposium on Software Testing and Analysis, Virtual Event, Denmark, July 11-17, 2021*, C. Cadar and X. Zhang, Eds. ACM, 2021, pp. 67–78. [Online]. Available: https://doi.org/10.1145/3460319.3464825

[10] Y. Lu, W. Sun, and M. Sun, "Towards mutation testing of reinforcement learning systems," *J. Syst. Archit.*, vol. 131, p. 102701, 2022. [Online]. Available: https://doi.org/10.1016/j.sysarc.2022.102701

[11] L.-C. Feng, X.-Y. Wang, S.-Y. Zhang, R.-Z. Gao, and Z.-H. Zhao, "Mutation operator reduction for cost-effective deep learning software testing via decision boundary change measurement," *Journal of Internet Technology*, vol. 23, no. 3, pp. 601–610, 2022. [Online]. Available: https://jit.ndhu.edu.tw/article/view/2705

[12] Y. Wang, Z. Zhang, Y. Yao, and Z. Huang, "A fine-grained evaluation of mutation operators for deep learning systems: A selective mutation approach," in *Proceedings of the 14th Asia-Pacific Symposium on Internetware, Internetware 2023, Hangzhou, China, August 4-6, 2023*, H. Mei, J. Lv, Z. Jin, X. Li, X. Yang, and X. Xia, Eds. ACM, 2023, pp. 123–133. [Online]. Available: https://doi.org/10.1145/3609437.3609453

[13] A. Ghanbari, D. Thomas, M. A. Arshad, and H. Rajan, "Mutation-based fault localization of deep neural networks," in *38th IEEE/ACM International Conference on Automated Software Engineering, ASE 2023, Luxembourg, September 11-15, 2023*. IEEE, 2023, pp. 1301–1313. [Online]. Available: https://doi.org/10.1109/ASE56229.2023.00171

[14] Y. Li, W. Shen, T. Wu, L. Chen, D. Wu, Y. Zhou, and B. Xu, "How higher order mutant testing performs for deep learning models: A fine-grained evaluation of test effectiveness and efficiency improved from second-order mutant-classification tuples," *Inf. Softw. Technol.*, vol. 150, p. 106954, 2022. [Online]. Available: https://doi.org/10.1016/j.infsof.2022.106954

[15] W. Shen, Y. Li, Y. Han, L. Chen, D. Wu, Y. Zhou, and B. Xu, "Boundary sampling to boost mutation testing for deep learning models," *Inf. Softw. Technol.*, vol. 130, p. 106413, 2021. [Online]. Available: https://doi.org/10.1016/j.infsof.2020.106413

[16] A. Ghanbari, "Decomposition of deep neural networks into modules via mutation analysis," in *Proceedings of the 33rd ACM SIGSOFT International Symposium on Software Testing and Analysis, ISSTA 2024, Vienna, Austria, September 16-20, 2024*, M. Christakis and M. Pradel, Eds. ACM, 2024, pp. 1669–1681. [Online]. Available: https://doi.org/10.1145/3650212.3680390

[17] A. V. Pizzoleto, F. C. Ferrari, J. Offutt, L. Fernandes, and M. Ribeiro, "A systematic literature review of techniques and metrics to reduce the cost of mutation testing," *J. Syst. Softw.*, vol. 157, 2019. [Online]. Available: https://doi.org/10.1016/j.jss.2019.07.100

[18] M. P. Usaola and P. R. Mateo, "Mutation testing cost reduction techniques: A survey," *IEEE Softw.*, vol. 27, no. 3, pp. 80–86, 2010. [Online]. Available: https://doi.org/10.1109/MS.2010.79

[19] I. N. Sneddon, *Fourier transforms*. Courier Corporation, 1995.

[20] G. P. Tolstov, *Fourier series*. Courier Corporation, 2012.

[21] A. V. Oppenheim and R. W. Schafer, *Discrete-Time Signal Processing*, 3rd ed. USA: Prentice Hall Press, 2009.

[22] A. Ghanbari and S. Tavakkol, "Using fourier analysis and mutant clustering to accelerate dnn mutation testing (replication package)," 2025. [Online]. Available: https://doi.org/10.5281/zenodo.17254335

[23] M. Papadakis, M. Kintis, J. Zhang, Y. Jia, Y. L. Traon, and M. Harman, "Chapter six - mutation testing advances: An analysis and survey," *Adv. Comput.*, vol. 112, pp. 275–378, 2019. [Online]. Available: https://doi.org/10.1016/bs.adcom.2018.03.015

[24] Y. Jia and M. Harman, "An analysis and survey of the development of mutation testing," *IEEE Trans. Software Eng.*, vol. 37, no. 5, pp. 649–678, 2011. [Online]. Available: https://doi.org/10.1109/TSE.2010.62

[25] J. Wang, G. Dong, J. Sun, X. Wang, and P. Zhang, "Adversarial sample detection for deep neural network through model mutation testing," in *Proceedings of the 41st International Conference on Software Engineering, ICSE 2019, Montreal, QC, Canada, May 25-31, 2019*, J. M. Atlee, T. Bultan, and J. Whittle, Eds. IEEE / ACM, 2019, pp. 1245–1256. [Online]. Available: https://doi.org/10.1109/ICSE.2019.00126

[26] Q. Hu, Y. Guo, M. Cordy, M. Papadakis, and Y. L. Traon, "MUTEN: mutant-based ensembles for boosting gradient-based adversarial attack," in *38th IEEE/ACM International Conference on Automated Software Engineering, ASE 2023, Luxembourg, September 11-15, 2023*. IEEE, 2023, pp. 1708–1712. [Online]. Available: https://doi.org/10.1109/ASE56229.2023.00042

[27] Q. Hu, L. Ma, X. Xie, B. Yu, Y. Liu, and J. Zhao, "Deepmutation++: A mutation testing framework for deep learning systems," in *34th IEEE/ACM International Conference on Automated Software Engineering, ASE 2019, San Diego, CA, USA, November 11-15, 2019*. IEEE, 2019, pp. 1158–1161. [Online]. Available: https://doi.org/10.1109/ASE.2019.00126

[28] R. Lin, Q. Zhou, B. Wu, and X. Nan, "Robustness evaluation for deep neural networks via mutation decision boundaries analysis," *Inf. Sci.*, vol. 601, pp. 147–161, 2022. [Online]. Available: https://doi.org/10.1016/j.ins.2022.04.020

[29] Z. Wang, H. You, J. Chen, Y. Zhang, X. Dong, and W. Zhang, "Prioritizing test inputs for deep neural networks via mutation analysis," in *43rd IEEE/ACM International Conference on Software Engineering, ICSE 2021, Madrid, Spain, 22-30 May 2021*. IEEE, 2021, pp. 397–409. [Online]. Available: https://doi.org/10.1109/ICSE43902.2021.00046

[30] Q. Hu, Y. Guo, X. Xie, M. Cordy, M. Papadakis, L. Ma, and Y. L. Traon, "Aries: Efficient testing of deep neural networks via labeling-free accuracy estimation," in *45th IEEE/ACM International Conference on Software Engineering, ICSE 2023, Melbourne, Australia, May 14-20, 2023*. IEEE, 2023, pp. 1776–1787. [Online]. Available: https://doi.org/10.1109/ICSE48619.2023.00152

[31] J. Sohn, S. Kang, and S. Yoo, "Arachne: Search-based repair of deep neural networks," *ACM Trans. Softw. Eng. Methodol.*, vol. 32, no. 4, pp. 85:1–85:26, 2023. [Online]. Available: https://doi.org/10.1145/3563210

[32] H. Wu, Z. Li, Z. Cui, and J. Liu, "Genmunn: A mutation-based approach to repair deep neural network models," *Int. J. Model. Simul. Sci. Comput.*, vol. 13, no. 2, pp. 2 341 008:1–2 341 008:17, 2022. [Online]. Available: https://doi.org/10.1142/S1793962323410088

[33] V. Riccio, N. Humbatova, G. Jahangirova, and P. Tonella, "Deepmetis: Augmenting a deep learning test set to increase its mutation score," in *36th IEEE/ACM International Conference on Automated Software Engineering, ASE 2021, Melbourne, Australia, November 15-19, 2021*. IEEE, 2021, pp. 355–367. [Online]. Available: https://doi.org/10.1109/ASE51524.2021.9678764

[34] H. Deokuliar, R. S. Sangwan, Y. Badr, and S. M. Srinivasan, "Improving testing of deep-learning systems," *Commun. ACM*, vol. 67, no. 3, pp. 44–48, 2024. [Online]. Available: https://doi.org/10.1145/3633311

[35] T. Zohdinasab, V. Riccio, and P. Tonella, "Focused test generation for autonomous driving systems," *ACM Trans. Softw. Eng. Methodol.*, vol. 33, no. 6, p. 152, 2024. [Online]. Available: https://doi.org/10.1145/3664605

[36] G. Jahangirova and P. Tonella, "An empirical evaluation of mutation operators for deep learning systems," in *13th IEEE International Conference on Software Testing, Validation and Verification, ICST 2020, Porto, Portugal, October 24-28, 2020*. IEEE, 2020, pp. 74–84. [Online]. Available: https://doi.org/10.1109/ICST46399.2020.00018

[37] L. Lyons and A. Ghanbari, "On accelerating deep neural network mutation analysis by neuron and mutant clustering," in *IEEE Conference on Software Testing, Verification and Validation, ICST 2025, Napoli, Italy, March 31 - April 4, 2025*. IEEE, 2025, pp. 267–278. [Online]. Available: https://doi.org/10.1109/ICST62969.2025.10989014

[38] M. Raghu, J. Gilmer, J. Yosinski, and J. Sohl-Dickstein, "SVCCA: singular vector canonical correlation analysis for deep learning dynamics and interpretability," in *Annual Conference on Neural Information Processing Systems*, I. Guyon, U. von Luxburg, S. Bengio, H. M. Wallach, R. Fergus, S. V. N. Vishwanathan, and R. Garnett, Eds., 2017, pp. 6076–6085.

[39] S. Lin, N. Liu, M. Nazemi, H. Li, C. Ding, Y. Wang, and M. Pedram, "Fft-based deep learning deployment in embedded systems," in *2018 Design, Automation & Test in Europe Conference & Exhibition, DATE 2018, Dresden, Germany, March 19-23, 2018*, J. Madsen and A. K. Coskun, Eds. IEEE, 2018, pp. 1045–1050. [Online]. Available: https://doi.org/10.23919/DATE.2018.8342166

[40] A. Lavin and S. Gray, "Fast algorithms for convolutional neural networks," in *2016 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2016, Las Vegas, NV, USA, June 27-30, 2016*. IEEE Computer Society, 2016, pp. 4013–4021. [Online]. Available: https://doi.org/10.1109/CVPR.2016.435

[41] S. Kullback and R. A. Leibler, "On information and sufficiency," *The Annals of Mathematical Statistics*, vol. 22, no. 1, pp. 79–86, 1951. [Online]. Available: http://www.jstor.org/stable/2236703

[42] J. Lin, "Divergence measures based on the shannon entropy," *IEEE Trans. Inf. Theory*, vol. 37, no. 1, pp. 145–151, 1991. [Online]. Available: https://doi.org/10.1109/18.61115

[43] M. Klabunde, T. Schumacher, M. Strohmaier, and F. Lemmerich, "Similarity of neural network models: A survey of functional and representational measures," *ACM Comput. Surv.*, vol. 57, no. 9, pp. 242:1–242:52, 2025. [Online]. Available: https://doi.org/10.1145/3728458

[44] D. Mitrović, M. Zeppelzauer, and C. Breiteneder, "Chapter 3 - features for content-based audio retrieval," in *Adv. in Computers: Improving the Web*, ser. Adv. in Comput. Elsevier, 2010, vol. 78, pp. 71–150.

[45] T. Kinnunen and H. Li, "An overview of text-independent speaker recognition: From features to supervectors," *Speech Communication*, vol. 52, no. 1, pp. 12–40, 2010.

[46] L. Dhulipala, D. Eisenstat, J. Lacki, V. Mirrokni, and J. Shi, "Hierarchical agglomerative graph clustering in poly-logarithmic depth," in *Annual Conference on Neural Information Processing Systems*, S. Koyejo, S. Mohamed, A. Agarwal, D. Belgrave, K. Cho, and A. Oh, Eds., 2022.

[47] S. Hussain, "Mutation clustering," Master's thesis, King's College London, 2008.

[48] C. Ji, Z. Chen, B. Xu, and Z. Zhao, "A novel method of mutation clustering based on domain analysis," in *Proceedings of the 21st International Conference on Software Engineering & Knowledge Engineering (SEKE'2009), Boston, Massachusetts, USA, July 1-3, 2009*. Knowledge Systems Institute Graduate School, 2009, pp. 422–425.

[49] M. Yu and Y. Ma, "Possibility of cost reduction by mutant clustering according to the clustering scope," *Softw. Test. Verification Reliab.*, vol. 29, no. 1-2, 2019. [Online]. Available: https://doi.org/10.1002/stvr.1692

[50] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proc. IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998. [Online]. Available: https://doi.org/10.1109/5.726791

[51] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *2016 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2016, Las Vegas, NV, USA, June 27-30, 2016*. IEEE Computer Society, 2016, pp. 770–778. [Online]. Available: https://doi.org/10.1109/CVPR.2016.90

[52] M. Sandler, A. G. Howard, M. Zhu, A. Zhmoginov, and L. Chen, "Mobilenetv2: Inverted residuals and linear bottlenecks," in *2018 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2018, Salt Lake City, UT, USA, June 18-22, 2018*. Computer Vision Foundation / IEEE Computer Society, 2018, pp. 4510–4520.

[53] M. Tan and Q. V. Le, "Efficientnet: Rethinking model scaling for convolutional neural networks," in *Proceedings of the 36th International Conference on Machine Learning, ICML 2019, 9-15 June 2019, Long Beach, California, USA*, ser. Proceedings of Machine Learning Research, K. Chaudhuri and R. Salakhutdinov, Eds., vol. 97. PMLR, 2019, pp. 6105–6114. [Online]. Available: http://proceedings.mlr.press/v97/tan19a.html

[54] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Annual Conference on Neural Information Processing Systems*, P. L. Bartlett, F. C. N. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, Eds., 2012, pp. 1106–1114.

[55] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," in *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, Y. Bengio and Y. LeCun, Eds., 2015. [Online]. Available: http://arxiv.org/abs/1409.1556

[56] L. Deng, "The MNIST database of handwritten digit images for machine learning research [best of the web]," *IEEE Signal Process. Mag.*, vol. 29, no. 6, pp. 141–142, 2012. [Online]. Available: https://doi.org/10.1109/MSP.2012.2211477

[57] H. Xiao, K. Rasul, and R. Vollgraf, "Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms," *CoRR*, vol. abs/1708.07747, 2017. [Online]. Available: http://arxiv.org/abs/1708.07747

[58] T. Clanuwat, M. Bober-Irizar, A. Kitamoto, A. Lamb, K. Yamamoto, and D. Ha, "Deep learning for classical japanese literature," *CoRR*, vol. abs/1812.01718, 2018. [Online]. Available: http://arxiv.org/abs/1812.01718

[59] Y. Netzer, T. Wang, A. Coates, A. Bissacco, B. Wu, and A. Y. Ng, "Reading digits in natural images with unsupervised feature learning," in *NIPS Workshop on Deep Learning and Unsupervised Feature Learning 2011*, 2011. [Online]. Available: http://ufldl.stanford.edu/housenumbers/nips2011_housenumbers.pdf

[60] A. Krizhevsky and G. Hinton, "Learning multiple layers of features from tiny images," University of Toronto, Toronto, Ontario, Tech. Rep. 0, 2009. [Online]. Available: https://www.cs.toronto.edu/~kriz/learning-features-2009-TR.pdf

[61] S. Lazebnik, C. Schmid, and J. Ponce, "Beyond bags of features: Spatial pyramid matching for recognizing natural scene categories," in *2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR 2006), 17-22 June 2006, New York, NY, USA*. IEEE Computer Society, 2006, pp. 2169–2178. [Online]. Available: https://doi.org/10.1109/CVPR.2006.68

[62] A. Khosla, N. Jayadevaprakash, B. Yao, and L. Fei-Fei, "Novel dataset for fine-grained image categorization," in *First Workshop on Fine-Grained Visual Categorization, IEEE Conference on Computer Vision and Pattern Recognition*, Colorado Springs, CO, June 2011.

[63] J. Deng, W. Dong, R. Socher, L. Li, K. Li, and L. Fei-Fei, "Imagenet: A large-scale hierarchical image database," in *2009 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR 2009), 20-25 June 2009, Miami, Florida, USA*. IEEE Computer Society, 2009, pp. 248–255. [Online]. Available: https://doi.org/10.1109/CVPR.2009.5206848

[64] D. Lewis, "Reuters-21578 Text Categorization Collection," UCI Machine Learning Repository, 1987, DOI: https://doi.org/10.24432/C52G6M.

[65] A. L. Maas, R. E. Daly, P. T. Pham, D. Huang, A. Y. Ng, and C. Potts, "Learning word vectors for sentiment analysis," in *Association for Computational Linguistics: Human Language Technologies*, D. Lin, Y. Matsumoto, and R. Mihalcea, Eds. The Association for Computer Linguistics, 2011, pp. 142–150.

[66] J. Liu and L. Song, "Second-order mutation testing cost reduction based on mutant clustering using SOM neural network model," in *IEEE 45th Annual Computers, Software, and Applications Conference, COMPSAC 2021, Madrid, Spain, July 12-16, 2021*. IEEE, 2021, pp. 974–979. [Online]. Available: https://doi.org/10.1109/COMPSAC51774.2021.00131

[67] C. Spearman, "The proof and measurement of association between two things," *International Journal of Epidemiology*, vol. 39, no. 5, pp. 1137–1150, 10 2010. [Online]. Available: https://doi.org/10.1093/ije/dyq191

[68] J. Cohen, *Statistical Power Analysis for the Behavioral Sciences*. Lawrence Erlbaum Associates, 1988.

[69] F. Tambon, F. Khomh, and G. Antoniol, "A probabilistic framework for mutation testing in deep neural networks," *Inf. Softw. Technol.*, vol. 155, p. 107129, 2023. [Online]. Available: https://doi.org/10.1016/j.infsof.2022.107129