Cluster Purge Loss: Structuring Transformer Embeddings for Equivalent Mutants Detection

Anonymous ACL submission

Abstract

Recent pre-trained transformer models achieve superior performance in various code processing objectives. However, although effective at optimizing decision boundaries, common approaches for fine-tuning them for downstream classification tasks — distance-based methods or training an additional classification head — often fail to thoroughly structure the embedding space to reflect nuanced intra-class semantic relationships. Equivalent code mutant detection is one of these tasks, where the quality of the embedding space is crucial to the performance of the models. We introduce a novel framework that integrates cross-entropy loss with a deep metric learning objective, termed Cluster Purge Loss. This objective, unlike conventional approaches, concentrates on adjusting fine-grained differences within each class, encouraging the separation of instances based on semantical equivalency to the class center using dynamically adjusted borders. Employing UniXCoder as the base model, our approach demonstrates state-of-the-art performance in the domain of equivalent mutant detection and produces a more interpretable embedding space.

1 Introduction

Since the introduction of the transformer architecture (Vaswani et al., 2017), large language models showed radical improvements on a great range of NLP (Raiaan et al., 2024) and code-related (Zheng et al., 2023) tasks. Moreover, LLMs can be further fine-tuned for downstream tasks using a domain-specific dataset(Parthasarathy et al., 2024). For bi-directional encoder transformers — ones excelling in analyzing existing code (Nijkamp et al., 2023) — the standard approach in fine-tuning is using a task-specific head to train with the rest of the transformer. However, for some such tasks requiring deep semantic understanding, the structure of the resulting embedding space is extremely important (Li et al., 2022), and the method above may struggle to provide it adequately. One such task is equivalent mutant detection (EMD).

Mutation testing (Jia and Harman, 2011) is a software testing approach. The principle of this approach is to generate programs based on an initial program under test by applying mutation operators. Such generated programs called *mutants*, are supposed to exhibit altered behavior so they can be used to examine the adequacy of test suits for that program. A mutant passing some test cases in a suit signifies the inability to catch a potential bug. Appendix D presents a graphic explanation and Appendix E shows examples of generated mutants. Although mutation testing is widely known and has applications in other fields of testing (e.g., test case prioritization (Lou et al., 2015), bug detection (Pradel and Sen, 2018), localization of faults (Papadakis and Le Traon, 2015)), one of the main reasons hindering its adoption is the existence of *equivalent mutants*. Such programs are semantically equal to their origin, thus producing the same output. Equivalent mutants have posed a persistent challenge, as their presence distorts test outcomes and the *mutation score*, which makes their detection necessary.

History of EMD includes a considerable number of different approaches such as constraint-based testing (Baer et al., 2020), compiler optimizations (Papadakis et al., 2015; Kintis et al., 2018) and machine learning, i.e. SVM (Naeem et al., 2020) and RNN based approaches (Peacock et al., 2021). A recent study by Tian et al. (2024) showed that LLMs significantly outperform previous techniques' Precision, Recall, and F1-score, demonstrating an average 35.69% gain in the latter. Their approach achieved the highest values with BERT (Devlin et al., 2019)-based uniXCoder (Guo et al., 2022), utilizing graph-guided masked attention(GGMA) based on the representation of dependency relation between variables in the source code - Data

Flow(Guo et al., 2021).

We hypothesize that even though mutants descended from the same original program - *mutant class* - are clustered and separated in the embedding space from other classes, the subtle intra-class differences between *equivalent* and *non-equivalent* mutants are not adequately formed and captured by the fine-tuned LLMs and the classifier alone. A further hypothesis was put forward that such properties can be obtained by utilizing deep metric learning(DML) (Mohan et al., 2023) and, in turn, improve the classification of mutants. However, most DML approaches such as contrastive loss (Chopra et al., 2005), triplet loss (Schroff et al., 2015), proxyNCA++ (Teh et al., 2020) concentrate at the inter-class level, without explicitly structuring instances inside formed class clusters.

In our work, we confirm the hypothesis about the embedding space and propose an approach of carefully combining Cross-Entropy Loss from the classification head with a new loss function named *Cluster Purge Loss*. The idea of this function is that for each class, we update the Exponential Moving Average of all distances between equivalent mutants and their origin, do the same for non-equivalent mutants, and then try to push or pull mutants beyond the resulting average radius of their counterparts just enough to aid fine-tuning with distinguishing between them.

By conducting an ablation study, using the same LLM (uniXCoder), classifier head (RoBERTa classifier), training data, number of epochs, batch size and optimizer hyperparameters, we show that our method increases **precision(5.12%), recall(0.57%) and f1-score(2.24%)** compared to the highest results obtained by Tian et al. (2024).

Thus, the contributions of this paper can be summarized as follows:

- Introduced new Deep Metric Learning loss function, which aims not to organize classes of instances but to adjust semantic relationships inside each already formed cluster according to the given binary distinction.
- Showed that applying DML approach can be beneficial during fine-tuning a large language model for specific downstream tasks.
- Obtained results superior to SOTA in EMD while isolating the performance gains attributable to the proposed approach.

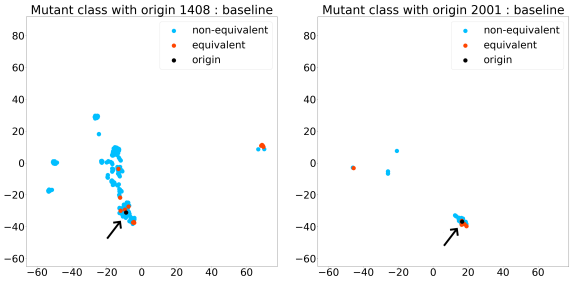

Figure 1: Baseline embeddings for classes with origins 1408 and 2001

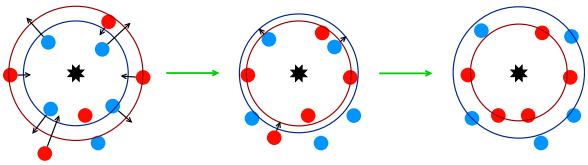

Figure 2: CPL conditions mutants to cross the EMA boundary of their counterparts, depicted as a circle of the opposite color, which is adjusted throughout training.

2 Proposed approach

Figure 1 illustrates a 2D t-SNE visualization of baseline model mutant embeddings of two classes generated by samples 1408 and 2001. In both cases, the distributions of equivalent and non-equivalent mutants within each class overlap significantly, and clustering fails to reflect semantic relationships. We hypothesize that introducing a secondary loss function explicitly designed to differentiate mutants in the embedding space based on their semantic equivalence to the ancestor program can facilitate the emergence of a more organized embedding space during fine-tuning. This improved organization, in turn, may enhance the performance of the classifier head by making distinctions between instances more straightforward.

The joint loss function is formulated as follows:

$$L = L_{\text{CPL}} \cdot \lambda + L_{\text{CE}} \tag{1}$$

Where L_{CE} is Cross-Entropy Loss obtained from the classifier head and L_{CPL} is the proposed Cluster Purge Loss. Combining loss functions may lead to a situation where their goals may be inconsistent (Luo et al., 2019). To combat this problem and balance the weight of each loss, the hyperparameter λ is used.

2.1 Cluster Purge Loss

We formalize the problem. Assume the minibatch size of m where each sample is:

$$(k_i, o_{k_i}, s_i, l_i) \text{ where } l_i \in \{0, 1\}, i \in \{0, \ldots, m\} \tag{2}$$

k_i is the unique identifier of a class, o_{k_i} is the embedding of the original program associated with k_i, s_i is the embedding of the another mutant belonging to k_i, and l_i represents its equivalence to the origin. Select all unique classes in the minibatch:

$$K = \{k_j \mid j \in \{0, \dots, m\}\} \tag{3}$$

For each unique class, find distances between its origin and equivalent mutants in the minibatch, where d_c^+ is a tuple of such distances for class c:

$$\forall c \in K, d_c^+ = (\text{dist}(o_{k_i}, s_i) \mid k_i = c \wedge l_i = 1, \\ i \in \{0, \dots, m\}) \tag{4}$$

The equation for exponential moving average, where γ is a *smoothing factor*:

$$\text{EMA}_{n+1} = \text{EMA}_n \cdot (1-s) + x \cdot s,\ s = \frac{2}{\gamma + 1} \tag{5}$$

Derive closed form for several $x_1, ..., x_h$:

$$\text{EMA}_{n+h} = \text{EMA}_n \cdot (1-s)^h + s \cdot \sum_{j=1}^{h} x_j \cdot (1-s)^{h-j} \tag{6}$$

Using eq.6 we can update EMA of distances from the origin to equivalent mutants for each class.The resulting average for the class c we will call a *positive verge* v_c^+. If v_c^+ is updated for the first time, then it is pre-initialized with the $d_{c\,0}^+$. Formulated as the following:

$$\forall c \in K, v_c^+ = 0 \implies v_c^+ = d_{c\,0}^+ \tag{7}$$

$$\forall c \in K, v_c^+ \leftarrow v_c^+ \cdot (1-s)^{|d_c^+|} + \\ s \cdot \sum_{j=1}^{|d_c^+|} d_{c\,j}^+ \cdot (1-s)^{|d_c^+|-j},\ s = \frac{2}{\gamma + 1} \tag{8}$$

Next, we carry out the same calculations for non-equivalent mutants to find a tuple of distances d_c^-(pairs with $l_i = 0$) and a *negative verge* v_c^-:

Finally, we can compute the loss function based on the current minibatch:

$$L_{\text{CPL}} = \frac{1}{m} \sum_{i=1}^{m} \Big(\Big[\text{dist}(o_{k_i}, s_i) - v_{k_i}^- + \zeta \Big]_+^{\alpha} \cdot l_i + \\ \Big[v_{k_i}^+ - \text{dist}(o_{k_i}, s_i) + \zeta \Big]_+^{\beta} \cdot (1 - l_i) \Big) \tag{9}$$

If $l_i = 1$, then s_i is equivalent and the calculation is as follows: distance from s_i to the origin o_{k_i} of its class k_i minus the negative verge for k_i and plus the margin ζ; then ReLU is applied and the resulting expression is raised to the power of α. Such formulation encourages keeping the distance of equivalent mutants to the origin less than the boundaries of non-equivalent mutants by ζ. The same principle applies if s_i is non-equivalent, but in the opposite direction and with the positive verge.

Hyperparameters α and β are introduced to control growth of the loss function for both cases separately, when asymmetric structuring is beneficial.

3 Experimentation

To assess our approach we conducted the ablation study. UniXCoder(110M) fine-tuned with GGMA and cross-entropy loss, which was found by Tian et al. (2024) to perform the best in terms of f1-score among LLMs and other approaches, was taken as the baseline of the study. It was compared with UnixCoder fine-tuned with the same setup modified to use a combination of cross-entropy and CPL. The altered setup inherited the values of all shared hyperparameters from the baseline, and its fine-tuning was based on the same dataset.

3.1 Dataset preparation

Dataset utilized in the baseline study was derived from MutantBench(van Hijfte and Oprescu, 2021) aggregating many previously published datasets. Tian et al. (2024) preprocessed it and obtained 3302 pairs of java mutants with the same origin method and accompanied with the equivalency label. $train_{base}$ of size 1652 was constructed by sampling 50/50 split of equivalent and non-equivalent mutant pairs, while $test_{base}$ was created with the rest totaling 1650 mutants.

During preprocessing, we determined the origins of all mutants in the datasets and based on them introduced 52 mutant classes, each assigned a sequential id. To construct $train_{cpl}$, we augmented each pair of mutants from $train_{base}$ with the class id, resulting in 1590 pairs. The same operation was done to create $test_{cpl}$ with 1580 pairs. The number of pairs in the obtained datasets is slightly lower due to duplicates being removed.

3.2 Implementation

$model_{base}$ is the pre-trained UnixCoder paired with the RoBERTa classification head. During fine-tuning the input sequence is constructed from the

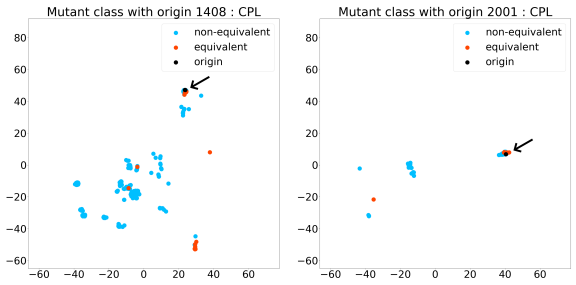

Figure 3: Embeddings after Cross-Entropy and Cluster Purge Loss

source code of each mutant and Data Flow Graph. The input sequence is converted into input vectors, following Guo et al. (2021), and fed to the forward pass method. Input token embeddings and GGMA matrix are calculated and passed to the UniXCoder in the encoder mode.The embedding representing a mutant is acquired by taking a normalized CLS token out of the last layer output. Subsequently, pairs of embeddings are passed to the classifier head and softmax. Finally, the cross-entropy loss is computed using equivalence probability labels.

For $model_{cpl}$, we set up the similar pipeline following Guo et al. and Tian et al. Features are extracted taking into account the addition of class id in $train_{cpl}$ and $model_{cpl}$ is implemented to include the calculation of CPL. The Model class stores verges in the buffer during fine-tuning, preserving them between epoches, and computes CPL and the final loss as described in subsection 2.1.

3.3 Evaluation

3.3.1 Experiment results

We conducted 56 experiments by fine-tuning $model_{cpl}$ on $train_{cpl}$ and evaluating on $test_{cpl}$ with $dist$ being normalized cosine distance, $\gamma = 12$, $\alpha = 2$, $\beta = 1/2$, $\zeta \in [-0.06, 0.01]$ with a step 0.01 and $\lambda \in [1.00, 1.30]$ with a step 0.05. Rationale of choosing hyperparameters is present in Appendix A. The obtained metrics were compared against $model_{base}$ fine-tuned with the same number of epochs = 30, batches = 4 and other shared hyperparameters. The results for all combinations of λ and ζ are presented in Appendix F where acquired precision(P), recall(R) and f1-score(F1) are stated. The best result in terms of f1-score is (P: 95.31%, R: 85.41%, F1: 89.46%) at λ = 1.15 and ζ = -0.05. Given that the metrics of $model_{base}$ is (P: 90.19%, R: 84.84%, F1: 87.22%), the absolute gain is (P: **5.12%**, R: **0.57%**, F1: **2.24%**). In Table 1, we also include results obtained for $model_{base}$ by Tian et al. using epochs = 10. It is clear that our approach shows better results for all metrics.

Technique	Precision	Recall	F1-score
$model_{CPL}$	**95.31%**	**85.41%**	**89.46%**
$model_{base}$	90.19%	84.84%	87.22%
$model_{base}$, Tian	94.33%	81.81%	86.58%

Table 1: Comparison with the baseline

3.3.2 Impact on embeddings distribution

To prove the hypothesis about Cluster Purge Loss promoting more organized embedding space, which is beneficial for EMD, the embeddings of mutants with origin 1408 and 2001 were extracted from the best performing $model_{cpl}$ and plotted after applying T-SNE(Figure 3). Non-equivalent mutants can be observed to be distributed significantly further away from the origin, while the distance to equivalent mutants varies. For the origin 1408, 2 clusters of equivalent mutants were formed, the first one is close to the origin, while the second is distanced from it. The latter can be explained by the negative ζ as discussed in Appendix A.

However, T-SNE doesn't always preserve global structure well. To investigate observations, the mean distance of embeddings of all non-equivalent mutants to their origin was computed: 0.105 ± 0.133 for $model_{base}$ and 0.398 ± 303 for $model_{cpl}$ with p < 0.0001. For all equivalent mutants: 0.111 ± 0.215 for $model_{base}$ and 0.189 ± 0.284 for $model_{cpl}$ with p = 0.83. That means that the ratio between the mean distance of non-equivalents to the origin and the mean distance of equivalents to the origin increased from 0.95 to 2.11 and is attributed to the statistically significant change in the distribution of the non-equivalent mutants.

Thus, we can conclude that our hypothesis holds and the introduction of CPL improved the performance on the equivalent mutant detection task by promoting the semantic meaning on distances between embeddings in the intra-class context.

4 Conclusion

In this study we introduced new Deep Metric Learning loss function named Cluster Purge Loss which organizes instances in already formed class clusters based on the semantical similarity to the class center. By the ablation study, we showed that using CPL in the joint loss formulation with cross-entropy loss shows state-of-the-art performance in equivalent mutant detection and found out that it is attributed to CPL impact on the embedding space.

5 Limitations

The first limitation of our work concerns the dataset used. For a fair comparison, we employed the same mutant pairs as the baseline study, forming $train_{cpl}$ with 1590 samples and $test_{cpl}$ with 1580 samples. The relatively small dataset size can affect the variance of fine-tuning results. Moreover, since all mutants are written in Java, it remains unclear how well our findings generalize to other programming languages.

The second limitation is that we ran only one trial for each of the 52 hyperparameter experiments due to limited computational resources. Conducting multiple runs for each experiment would help reduce variance caused by randomness and produce more robust conclusions.

References

Michael Baer, Norbert Oster, and Michael Philippsen. 2020. MutantDistiller: Using Symbolic Execution for Automatic Detection of Equivalent Mutants and Generation of Mutant Killing Tests. In *2020 IEEE International Conference on Software Testing, Verification and Validation Workshops (ICSTW)*, pages 294–303, Porto, Portugal. IEEE.

S. Chopra, R. Hadsell, and Y. LeCun. 2005. Learning a similarity metric discriminatively, with application to face verification. In *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05)*, volume 1, pages 539–546 vol. 1.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. In *Proceedings of the 2019 Conference of the North*, pages 4171–4186, Minneapolis, Minnesota. Association for Computational Linguistics.

Daya Guo, Shuai Lu, Nan Duan, Yanlin Wang, Ming Zhou, and Jian Yin. 2022. UniXcoder: Unified Cross-Modal Pre-training for Code Representation. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 7212–7225, Dublin, Ireland. Association for Computational Linguistics.

Daya Guo, Shuo Ren, Shuai Lu, Zhangyin Feng, Duyu Tang, Shujie Liu, Long Zhou, Nan Duan, Alexey Svyatkovskiy, Shengyu Fu, Michele Tufano, Shao Kun Deng, Colin B. Clement, Dawn Drain, Neel Sundaresan, Jian Yin, Daxin Jiang, and Ming Zhou. 2021. Graphcodebert: Pre-training code representations with data flow.

Yue Jia and Mark Harman. 2011. An Analysis and Survey of the Development of Mutation Testing. *IEEE Transactions on Software Engineering*, 37(5):649–678.

Marinos Kintis, Mike Papadakis, Yue Jia, Nicos Malevris, Yves Le Traon, and Mark Harman. 2018. Detecting Trivial Mutant Equivalences via Compiler Optimisations. *IEEE Transactions on Software Engineering*, 44(4):308–333.

Xiang Li, John Thickstun, Ishaan Gulrajani, Percy S Liang, and Tatsunori B Hashimoto. 2022. Diffusion-lm improves controllable text generation. In *Advances in Neural Information Processing Systems*, volume 35, pages 4328–4343. Curran Associates, Inc.

Yiling Lou, Dan Hao, and Lu Zhang. 2015. Mutation-based test-case prioritization in software evolution. In *2015 IEEE 26th International Symposium on Software Reliability Engineering (ISSRE)*, pages 46–57.

Hao Luo, Youzhi Gu, Xingyu Liao, Shenqi Lai, and Wei Jiang. 2019. Bag of tricks and a strong baseline for deep person re-identification. In *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, pages 1487–1495.

Deen Dayal Mohan, Bhavin Jawade, Srirangaraj Setlur, and Venu Govindaraju. 2023. Chapter 4 - deep metric learning for computer vision: A brief overview. In Venu Govindaraju, Arni S.R. Srinivasa Rao, and C.R. Rao, editors, *Deep Learning*, volume 48 of *Handbook of Statistics*, pages 59–79. Elsevier.

Muhammad Rashid Naeem, Tao Lin, Hamad Naeem, and Hailu Liu. 2020. A machine learning approach for classification of equivalent mutants. *Journal of Software: Evolution and Process*, 32(5):e2238. E2238 smr.2238.

Erik Nijkamp, Hiroaki Hayashi, Caiming Xiong, Silvio Savarese, and Yingbo Zhou. 2023. Codegen2: Lessons for training llms on programming and natural languages. *CoRR*, arXiv:2305.02309.

Mike Papadakis, Yue Jia, Mark Harman, and Yves Le Traon. 2015. Trivial compiler equivalence: A large scale empirical study of a simple, fast and effective equivalent mutant detection technique. In *2015 IEEE/ACM 37th IEEE International Conference on Software Engineering*, volume 1, pages 936–946.

Mike Papadakis and Yves Le Traon. 2015. Metallaxis-fl: mutation-based fault localization. *Softw. Test. Verif. Reliab.*, 25(5–7):605–628.

Venkatesh Balavadhani Parthasarathy, Ahtsham Zafar, Aafaq Khan, and Arsalan Shahid. 2024. The Ultimate Guide to Fine-Tuning LLMs from Basics to Breakthroughs: An Exhaustive Review of Technologies, Research, Best Practices, Applied Research Challenges and Opportunities. *arXiv preprint*. ArXiv:2408.13296.

Samuel Peacock, Lin Deng, Josh Dehlinger, and Suranjan Chakraborty. 2021. Automatic Equivalent Mutants Classification Using Abstract Syntax Tree Neural Networks. In *2021 IEEE International Conference on Software Testing, Verification and Validation Workshops (ICSTW)*, pages 13–18.

Michael Pradel and Koushik Sen. 2018. Deepbugs: a learning approach to name-based bug detection. *Proc. ACM Program. Lang.*, 2(OOPSLA).

Mohaimenul Azam Khan Raiaan, Md. Saddam Hossain Mukta, Kaniz Fatema, Nur Mohammad Fahad, Sadman Sakib, Most Marufatul Jannat Mim, Jubaer Ahmad, Mohammed Eunus Ali, and Sami Azam. 2024. A Review on Large Language Models: Architectures, Applications, Taxonomies, Open Issues and Challenges. *IEEE Access*, 12:26839–26874.

Florian Schroff, Dmitry Kalenichenko, and James Philbin. 2015. Facenet: A unified embedding for face recognition and clustering. In *IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2015, Boston, MA, USA, June 7-12, 2015*, pages 815–823. IEEE Computer Society.

Eu Wern Teh, Terrance DeVries, and Graham W. Taylor. 2020. ProxyNCA++: Revisiting and Revitalizing Proxy Neighborhood Component Analysis. In *Computer Vision – ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part XXIV*, pages 448–464, Berlin, Heidelberg. Springer-Verlag.

Zhao Tian, Honglin Shu, Dong Wang, Xuejie Cao, Yasutaka Kamei, and Junjie Chen. 2024. Large Language Models for Equivalent Mutant Detection: How Far Are We? *arXiv preprint*. ArXiv:2408.01760.

Lars van Hijfte and Ana Oprescu. 2021. Mutantbench: an equivalent mutant problem comparison framework. In *n 2021 IEEE International Conference on Software Testing, Verification and Validation Workshops (ICSTW). IEEE, 7–12*. IEEE.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Ł ukasz Kaiser, and Illia Polosukhin. 2017. Attention is All you Need. In *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc.

Zibin Zheng, Kaiwen Ning, Yanlin Wang, Jingwen Zhang, Dewu Zheng, Mingxi Ye, and Jiachi Chen. 2023. A Survey of Large Language Models for Code: Evolution, Benchmarking, and Future Trends. *arXiv preprint*.

A Hyperparameters selection

To evaluate our approach, we conducted a series of experiments aiming to explore the hyperparameter space of Cluster Purge Loss. As the distance function normalized cosine distance was chosen:

$$\text{dist}(a, b) = 1 - \frac{\text{cossim}(a, b)}{2} + 1 \qquad (10)$$

Inverse formulation means that the codomain is [0,1] where 0 indicates collinearity of vectors. The smoothing factor γ was chosen as 12 based on the preliminary experiments. The value of the exponent of a loss term for equivalent mutants α is 2 and the exponent of a loss term for non-equivalent mutants β is 1/2. Such initial values of α and β are based on the assumption that equivalent mutants are already located close enough to their origin, and to give semantic similarity properties to the embedding space, emphasis must be placed on changing the distribution of non-equivalent mutants. Since the square function shows sublinear growth on values close to 0 included in the codomain [0,1] of the distance function, and the root function, on the contrary, grows superlinearly, then the loss value for non-equivalent mutants will grow faster with the distance from the verge than for equivalent ones.

The margin ζ between mutants and the corresponding verges and the coefficient λ at L_{CPL} are considered the most influental and explored in ranges: $\zeta \in [-0.06, 0.01]$ with the step 0.01 and $\lambda \in [1.00, 1.30]$ with the step 0.05. Such intervals are chosen based on the preliminary findings showing that smaller values are more favorable. For λ we explain it by the assumption that Cluster Purge Loss is more beneficial in the setup as the lesser term in the equation shifts the negative gradient towards the more optimal solution by imposing the semantic meaning on the distance. For ζ, we assume that since the model's ability to capture semantic differences between mutants is imperfect, a negative boundary can create an "error zone" for those mutants that cannot be correctly ordered without worsening the arrangement of the rest.

B Scientific artifacts usage

Pre-trained UniXCoder model, $train_{base}$, $test_{base}$, $model_{base}$ are obtained from `https://github.com/tianzhaotju/EMD` where the replication package for Tian et al. (2024) was released. It was sanctioned for replication, future research, and practical use, which we consider our usage to fall under.

C Computational budget

Each of the 52 experiments aimed at hyperparameter search required approximately 2.25 GPU-hours on a single NVIDIA RTX 4060, yielding a total computational cost of around 126 hours.

D Mutation testing

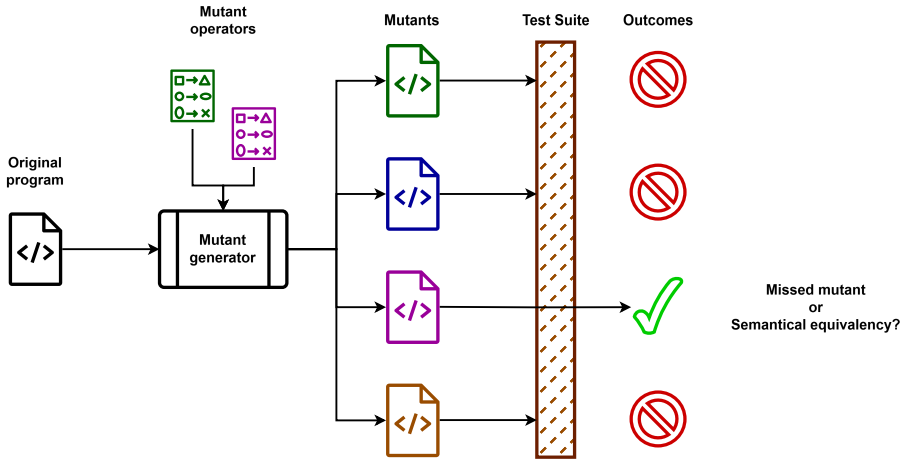

Figure 4: The process of mutation testing of the test suite for the original program. Equivalent mutants don't let make conclusions about the true mutation score.

E Mutants examples

Origin	Non-equivalent Mutant	Equivalent Mutant

1 `int binSearch(int arr[], int x) {`
2 `int l = 0;`
3 `int h = arr.length - 1;`
4 `while (l <= h) {`
5 `int mid = l + (h - l) / 2;`
6 `if (arr[mid] == x)`
7 `return mid;`
8 `if (arr[mid] < x)`
9 `l = mid + 1;`
10 `else`
11 `h = mid - 1;`
12 `}`
13 `return -1;`
14 `}`
	`int binSearch(int arr[], int x) {`
	`int l = 0;`
	`int h = arr.length - 1;`
	`while (l <= h) {`
	`int mid = l + (h - l) / 2;`
	`if (arr[mid++] == x)`
	`return mid;`
	`if (arr[mid] < x)`
	`l = mid + 1;`
	`else`
	`h = mid - 1;`
	`}`
	`return -1;`
	`}`
		`int binSearch(int arr[], int x) {`
		`int l = 0;`
		`int h = arr.length - 1;`
		`while (l <= h) {`
		`int mid = l + (h - l) / 2;`
		`if (arr[mid] == x)`
		`return mid++;`
		`if (arr[mid] < x)`
		`l = mid + 1;`
		`else`
		`h = mid - 1;`
		`}`
		`return -1;`
		`}`

Table 2: Examples of code mutants. The first column shows the origin method intended to perform binary search on array *arr* to find *x*. The second column is a non-equivalent mutant created by applying Unary Operator Insertion (UIO) to the line 6. Post-increment affects the return statement inside if clause resulting in returning the wrong value.The third column shows an equivalent mutant produced by applying UIO to the line 7. In this case post-increment doesn't influence behaviour as the method execution ends.

F Model performance matrix

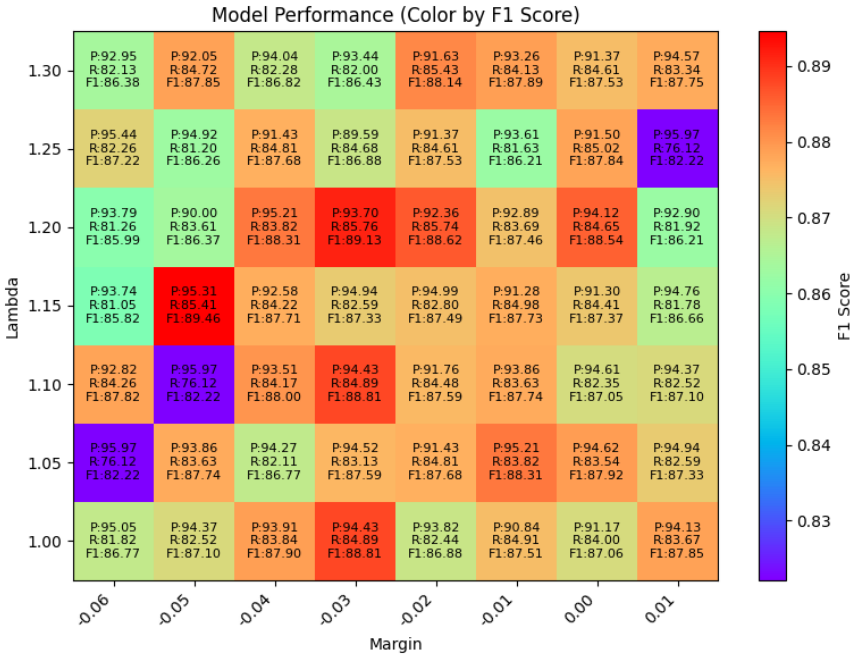

Figure 5: Model performance matrix presenting Precision(P), Recall(R), F1-score(F1) of the $model_{CPL}$ for different values of λ and ζ. Also, it can be observed that the matrix of metrics is heterogeneous, that can be attributed to the non-linear nature of interaction between λ and ζ.