Verifier Selection using LLMs

By Patrik Felbinger

This project was completed during Google Summer of Code 2025 in collaboration with LMU Munich. It addresses the challenge of selecting the most effective software verifier for a given C program, a central task in the annual Software Verification Competition (SV-COMP) [Beyer & Strejček, 2025]. Since no single verifier consistently outperforms others, predicting the most suitable verifier can reduce CPU and memory usage while improving verification quality.

To tackle this problem, we developed a machine-learning framework that predicts which verifier is most likely to succeed on a given program. The project produced two complementary components: Verifier-Moira, a modular library for verifier selection, and Verifier-Moira-Data, a companion repository for training and publishing predictors. Verifier-Moira allows users to choose an embedder, such as microsoft/graphcodebert-base, to represent C programs as vectors, and a predictor, such as LightGBM or XGBoost, trained on SV-COMP 2025 data to estimate verifier performance. The system outputs a ranked list of verifiers and can be easily extended with new embedders, predictors, or datasets.

Verifier-Moira-Data provides the training pipelines and pretrained predictors used by Verifier-Moira, simplifying the integration of new predictors and embedders. Predictors are trained on SV-COMP datasets, validated, and published for direct use. Evaluation against two baselines—the virtual best verifier as an upper bound and the best verifier-per-property as a lower bound—showed that trained predictors can consistently perform between these baselines, demonstrating improvements over static selection methods while remaining computationally efficient.

Training

We trained four types of predictors on the SV-COMP 2025 dataset[1] using Verifier-Moira and Verifier-Moira-Data:

- LightGBM (LGBM). A gradient boosting framework based on decision trees, optimized for efficiency in both memory and computation [Ke et al., 2017].
- XGBoost. An optimized gradient boosting library that incorporates regularization and advanced parallelization strategies, widely used in machine learning competitions [Chen & Guestrin, 2016].
- CatBoost. A gradient boosting algorithm designed to handle categorical features effectively and to reduce overfitting through ordered boosting [Dorogush et al., 2018].
- Neural Classifier. A feed-forward neural network classifier trained on embedding representations, serving as a deep learning baseline for comparison with boosting-based methods.

To evaluate predictor performance, we selected a diverse set of embedding models in terms of training domain and size that are summarized in Table 1. Codet5p (110M) and GraphCodeBERT (125M) are code-specialized, capturing syntactic and semantic program features, while MiniLM (33M) provides general-purpose text embeddings for contrast. Jina-embeddings-v2 (161M) covers multi-language code, and nomic-embed-text:v1.5 (137M) was trained on a mixture of text and code, bridging both domains. This set enables assessment across specialized, general, and hybrid embeddings of varying capacities.

[1] https://sv-comp.sosy-lab.org/2025/results/results-verified/

Table 1: Summary of embedding models used in this work, categorized by architecture type and parameter count (in millions).

Embedder	Type	Parameters (in M)
Salesforce/codet5p-110m-embedding[2]	Transformers	110
jinaai/jina-embeddings-v2-base-code[3]	Transformers	161
sentence-transformers/all-MiniLM-L12-v2[4]	Transformers	33
microsoft/graphcodebert-base[5]	Transformers	125
nomic-embed-text:v1.5[6]	Ollama	137

The predictors were trained on the SV-COMP 2025 results for a selected set of verifiers [Beyer & Strejček, 2025]. During evaluation, the verifier with the highest predicted score was chosen, and its actual outcome was retrieved from the SV-COMP dataset to assess prediction accuracy. To interpret the results, we used the official SV-COMP scoring scheme (Table 2), where correct verifications receive positive points, incorrect outcomes are penalized with larger negative scores, and missing results (e.g., timeouts or crashes) yield zero points.

Table 2: Adapted from https://sv-comp.sosy-lab.org/2025/rules.php

Points	Reported result	Description
0	UNKNOWN	Failure to compute verification result, out of resources, program crash.
+1	FALSE correct	The error in the program was found and a violation witness was confirmed.
−16	FALSE incorrect	An error is reported for a program that fulfills the specification (false alarm, incomplete analysis).
+2	TRUE correct	The program was analyzed to be free of errors and a correctness witness was confirmed.
−32	TRUE incorrect	The program had an error but the competition candidate did not find it (missed bug, unsound analysis).

Results

This section presents the performance of the trained predictors for each embedder. In addition to the listed predictors, a K-Nearest Neighbor (KNN, k=5) approach was applied for each embedder. For a given verification task, KNN selects the best verifier based on the five most similar software verification tasks. The models are trained only on the most relevant verifiers rather than the full set. To assess predictor performance, the following baselines were used:

- Virtually Best Verifier: The most optimal verifier selection for SV-COMP25 as an upper boundary if predictors were trained on all verifiers.

[2] https://huggingface.co/Salesforce/codet5p-110m-embedding
[3] https://huggingface.co/jinaai/jina-embeddings-v2-base-code
[4] https://huggingface.co/sentence-transformers/all-MiniLM-L12-v2
[5] https://huggingface.co/microsoft/graphcodebert-base
[6] https://ollama.com/library/nomic-embed-text:v1.5

- Virtually Best Verifier (Filtered): The most optimal verifier selection for SV-COMP25 given a predefined selection of verifiers that are used in training, therefore serving as the real upper boundary for the predictor performance.
- Best Verifier per Property: This selects the best verifier for each property given the SV-COMP25 results. This serves as a medium boundary and should be beaten by predictors.
- Random: A random verifier selector given the filter of verifiers. This serves as the lowest boundary to evaluate low performing predictors.

Average Score per Predictor

Figure 1 shows the average score of each predictor, with baselines indicated as vertical lines in the bar plot. Among the predictors, the neural classifiers, XGBoost, and LGBM achieve the strongest performance, consistently outperforming the *Best Verifier per Property* baseline, though still falling short of the *Virtually Best Verifier (filtered)*—the upper bound given the verifiers used during training.

The neural classifier with the *jinaai* embedder achieves the highest overall score. XGBoost follows closely, showing more stable performance across embedders and generally surpassing LGBM. KNN ranks next, with slightly higher scores than the *Best Verifier per Property* baseline. By contrast, the current CatBoost configuration underperforms compared to the *Best Verifier per Property* but still exceeds the random baseline, indicating that the model is either insufficiently trained or not well-suited to the task.

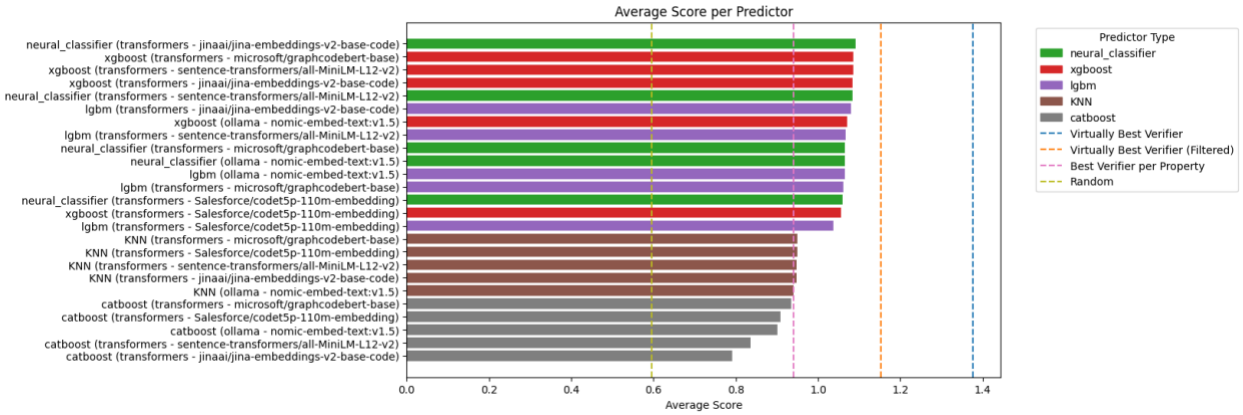

Figure 1: Average score per predictor and embedder. Baselines are vertical lines.

Average Score in relation to CPU Time

To provide a broader view of performance, Figure 2 compares the average score with the average CPU time, while bubble size indicates memory consumption (larger bubbles correspond to higher memory usage). The vertical and horizontal reference lines mark the performance of the *Best Verifier per Property*. As in Figure 1, predictors that lie above the horizontal line outperform this baseline. Points to the left of the vertical line indicate lower CPU time, and these are often associated with smaller bubbles, suggesting reduced memory usage. A clear overall trend emerges: higher scores generally coincide with lower CPU time and lower memory consumption. The neural classifier requires slightly more CPU time because it was trained solely to predict scores, without incorporating any information about CPU or memory usage, whereas the other predictors included a small CPU and memory component in their scoring.

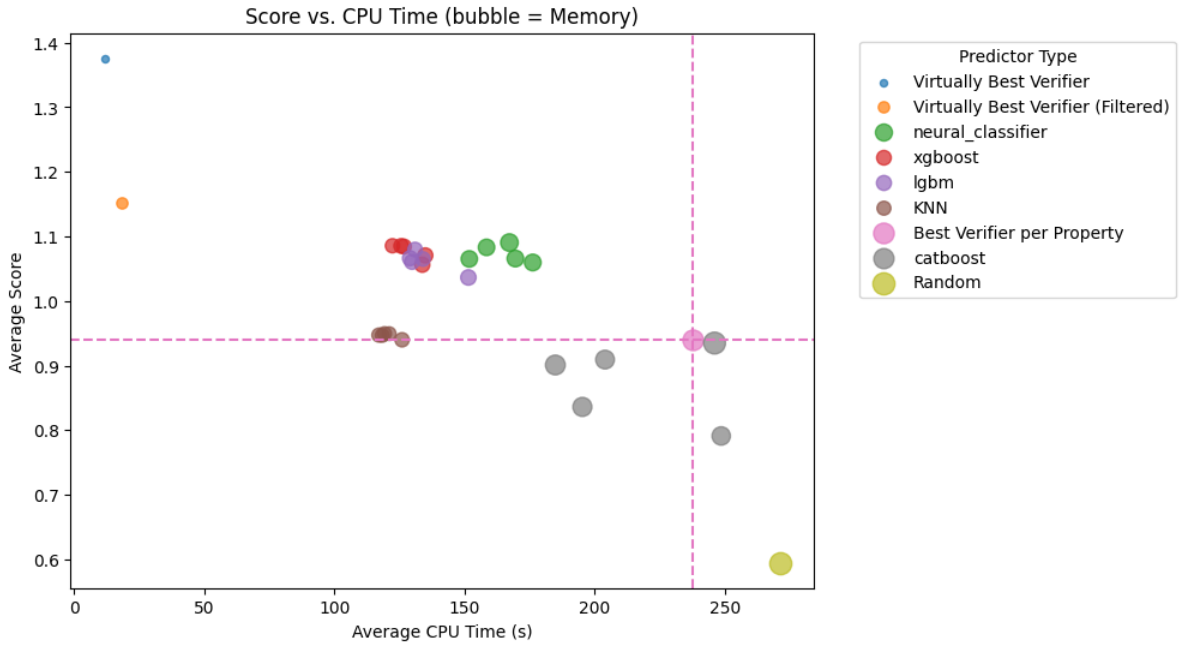

Figure 2: Average Score in relation to the average CPU time (s).

Score in relation to Embedder Size

The following figure illustrates predictor performance in relation to embedder size, measured by the number of parameters. Notably, embedder size appears to have little to no impact on predictor performance. Future studies should explore larger embedders, such as Qwen3 0.6B which offers an extended context window. In addition, the effect of verbose .i input files should be investigated, as it might introduce noise into embeddings.

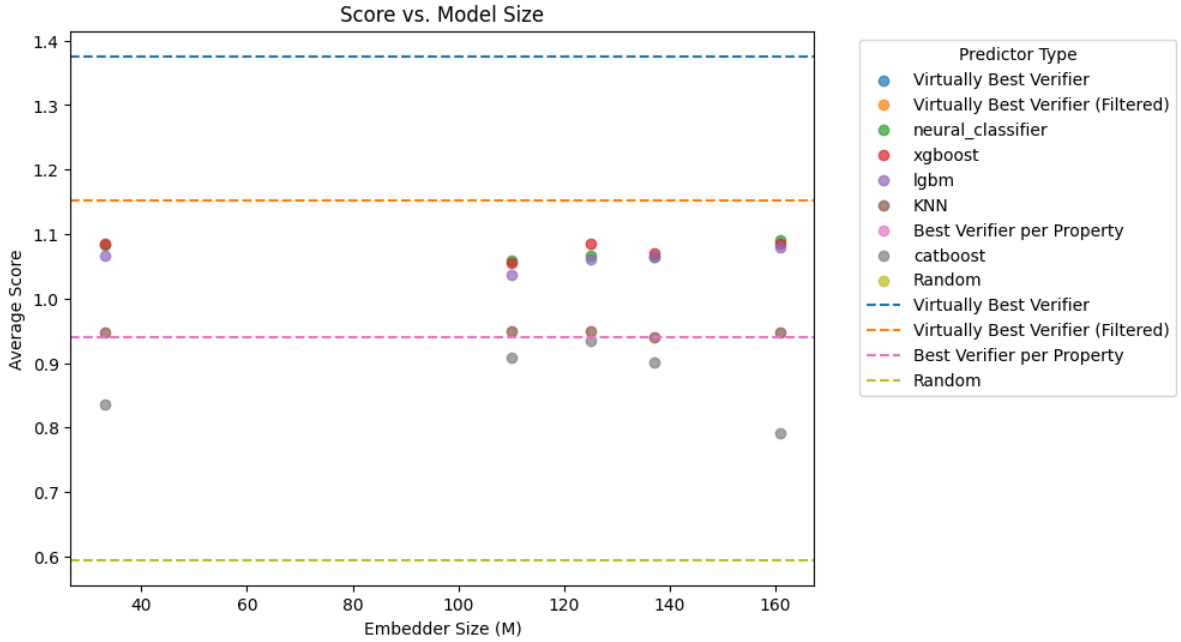

Figure 3: Average Score in relation to Embedder Size

Influence of Predictor Size

The capability of a predictor can be assessed by its size. Since models of different predictor types cannot easily be compared, a proxy metric is introduced – the Predictor Size. It is the file size of the *model.txt*, the serialized model content, and for neural classifier the file size of *model.onnx*. The following figure depicts this relation. The catboost models on the other hand are larger but less capable at the same time. XGBoost models have more reliable performance on various embedders but are larger than LightGBM. Neural Classifiers offer strong performance with little predictor size. It should be noted that those sizes depend on training setup and therefore should not be used to evaluate the capabilities of a predictor type.

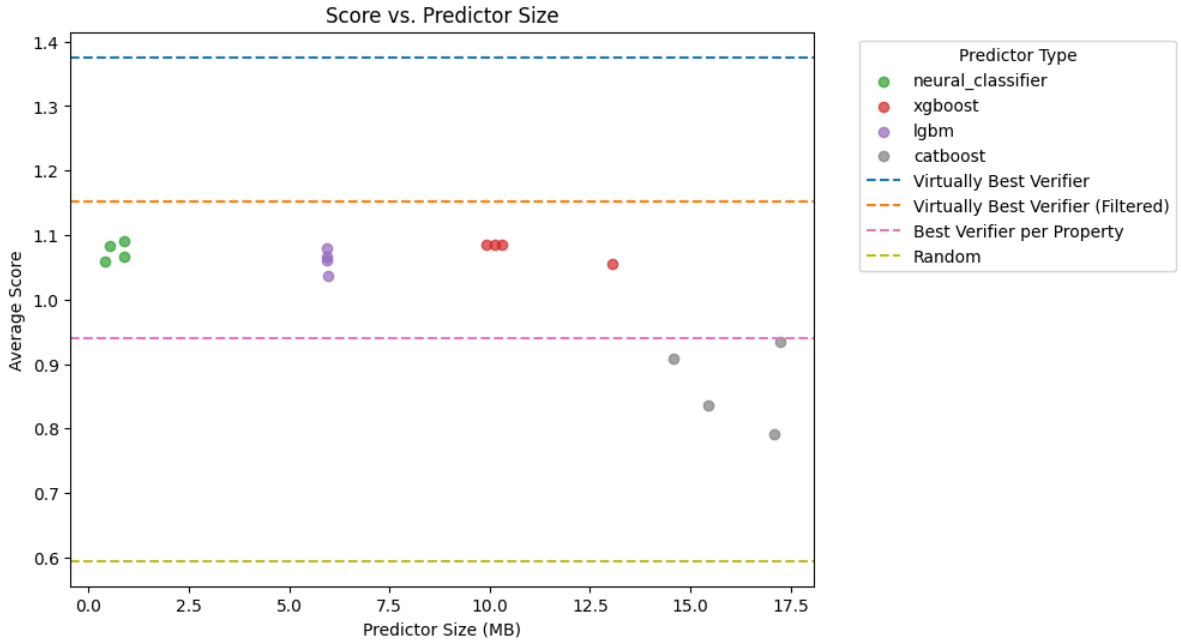

Figure 4: Average score in relation to predictor size

Predictor Aggregation Table

The overall performance of each predictor type is summarized in Table 3.

Table 3. Performance comparison of predictor types based on total verification score, average CPU time, memory consumption, and model size.

Predictor Type	Total Score	Avg. CPU Time (s)	Avg. Memory (MB)	Size (MB)
Virtually Best Verifier	9243	11	194	-
Virtually Best Verifier (Filtered)	7743	18	452	-
xgboost	7232.4	128	777	10.84
neural_classifier	7208.4	164	1007	0.68
lgbm	7132.8	135	794	5.94
KNN	6360	120	692	-
Best Verifier per Property	6320	237	1526	-
catboost	5876	215	1374	16.08
Random	3997	271	1749	-

All trained predictors, except CatBoost, outperform the Best Verifier per Property baseline. XGBoost and the Neural Classifier achieve the highest scores (~7200), approaching the Virtually Best Verifier (Filtered) upper bound. LGBM performs slightly weaker but remains competitive, while KNN provides moderate accuracy with lower CPU and memory usage. CatBoost underperforms in both accuracy and efficiency, indicating limitations in its current setup. Overall, XGBoost offers the best balance of performance and resource usage, whereas the Neural Classifier achieves similar accuracy at a higher computational cost.

Conclusion

This project demonstrates that machine-learning predictors can effectively support verifier selection for C programs in the SV-COMP setting. By leveraging diverse embedding models—ranging from code-specialized transformers to general-purpose and hybrid text-code embeddings—and training predictors such as XGBoost, LightGBM, CatBoost, and neural classifiers on SV-COMP 2025 data, Verifier-Moira consistently outperforms the best verifier per property while remaining computationally efficient. Evaluation shows that neural classifiers and XGBoost achieve the highest verification scores, closely approaching the upper bound defined by the Virtually Best Verifier (Filtered), while smaller or less optimized predictors still surpass naive baselines. These results highlight the use of embedders and careful predictor design. Future work may explore larger embeddings, enhanced preprocessing, and neural regressors to further improve prediction accuracy and resource efficiency.

Future Work

Future extensions may include the use of larger and more diverse embedders, the exploration of neural regressors, and improved preprocessing to clean .i files, which sometimes introduce noise into embeddings. Additional improvements in error handling and version validation for predictors would also further strengthen the robustness of the system.

Links

Verifier-Moira: https://gitlab.com/sosy-lab/software/verifier-moira
Verifier-Moira-Data: https://gitlab.com/sosy-lab/research/data/verifier-moira-data

Acknowledgements

I gratefully acknowledge the guidance of Marian Lingsch-Rosenfeld, whose expertise and mentorship were essential in shaping this project into a complete end-to-end workflow. His support ensured that both libraries could be successfully realized in a single summer, and I look forward to seeing them utilized in the upcoming SV-COMP 2026.

References

1. Chen, T., & Guestrin, C. (2016). *XGBoost: A Scalable Tree Boosting System.* In Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD '16) (pp. 785–794). Association for Computing Machinery. https://doi.org/10.1145/2939672.2939785
2. Ke, G., Meng, Q., Finley, T., Wang, T., Chen, W., Ma, W., Ye, Q., & Liu, T.-Y. (2017). *LightGBM: A Highly Efficient Gradient Boosting Decision Tree.* In Advances in Neural Information Processing Systems (NeurIPS 30), 3146–3154.
3. Dorogush, A. V., Ershov, V., & Gulin, A. (2018). *CatBoost: Gradient Boosting with Categorical Features Support.* arXiv:1810.11363. https://arxiv.org/abs/1810.11363
4. Beyer, D., Strejček, J. (2025). Improvements in Software Verification and Witness Validation: SV-COMP 2025. In: Gurfinkel, A., Heule, M. (eds) Tools and Algorithms for the Construction and Analysis of Systems. TACAS 2025. Lecture Notes in Computer Science, vol 15698. Springer, Cham. https://doi.org/10.1007/978-3-031-90660-2_9