

Using Ensemble Inference to Improve Recall of Clone Detection

Gul Aftab Ahmed
Lero Research Centre
Trinity College Dublin
Dublin, Ireland
ahmedga@tcd.ie

James Vincent Patten
Dept. of Computer Science and
Information Systems
University of Limerick
Limerick, Ireland
james.patten@lero.ie

Yuanhua Han
WN Digital IPD and
Trustworthiness Enabling
Huawei Technologies Co., Ltd.
Xi'an, Shaanxi, China
hanyuanhua2@huawei.com

Guoxian Lu
WN Digital IPD and
Trustworthiness Enabling
Huawei Technologies Co., Ltd.
Shanghai, China
luguoxian@huawei.com

David Gregg
Lero Research Centre
Trinity College Dublin
Dublin, Ireland
david.gregg@tcd.ie

Jim Buckley
Dept. of Computer Science and
Information Systems
University of Limerick
Limerick, Ireland
jim.buckley@ul.ie

Muslim Chochlov
Dept. of Computer Science and
Information Systems
University of Limerick
Limerick, Ireland
muslim.chochlov@ul.ie

Abstract—Large-scale source-code clone detection is a challenging task. In our previous work, we proposed an approach (SSCD) that leverages artificial neural networks and approximates nearest neighbour search to effectively and efficiently locate clones in large-scale bodies of code, in a time-efficient manner. However, our literature review suggests that the relative efficacy of differing neural network models has not been assessed in the context of large-scale clone detection approaches. In this work, we aim to assess several such models individually, in terms of their potential to maximize recall, while preserving a high level of precision during clone detection. We investigate if ensemble inference (in this case, using the results of more than one of these neural network models in combination) can further assist in this task.

To assess this, we employed four state-of-the-art neural network models and evaluated them individually/in combination. The results, on an illustrative dataset of approximately 500K lines of C/C++ code, suggest that ensemble inference outperforms individual models in all trialled cases, when recall is concerned. Of individual models, the ADA model (belonging to the ChatGPT family of models) has the best performance. However commercial companies may not be prepared to hand their proprietary source code over to the cloud, as required by that approach. Consequently, they may be more interested in an ensemble-combination of CodeBERT-based and CodeT5 models, resulting in similar (if slightly lesser) recall and precision results.

Index Terms—clone detection, artificial neural networks, ensemble inference

NOTE TO PRACTITIONERS

This is the accepted version of the paper submitted to the International Workshop on Software Clones (IWSC 2023). The final version should be accessible at <https://doi.org/10.1109/IWSC60764.2023.00010>.

I. INTRODUCTION

Clone detection is the process of locating textually or functionally exact or similar pieces of code [1]. The reasons

for performing clone detection vary, but can involve the need to address software maintenance issues (where, for example, a vulnerability found in the clone-source should be alerted to the clone-destination) and licensing issues (where code from one licensing regime has historically, and inappropriately, been cloned into a system with another licensing regime) among other tasks.

Locating clones manually in large pieces of code is effort-intensive and so, over the past few decades, many clone detection approaches and their implementing tools were proposed, showing mixed results [1], [2], particularly with respect to type three and type four clones where the textual-code match is less exact and so techniques need to be more sophisticated [1].

More recently, artificial intelligence (AI) models were utilized and reported state-of-the-art results in pairwise clone detection tasks [3], [4]. However, these AI-based approaches do not address clone detection at scale, which is often needed in real-world settings. This is because the pair-wise comparison of code-segments results in combinatorial effort, as the scale of the systems involved grows.

In our previous work, a novel approach (SSCD) was proposed that addresses this issue: it allows for AI-based, clone detection at scale, based on a global, nearest-neighbour comparison of code-segment vectors encoded by a fine-tuned, CodeBERT-based ANN [5]. When applied to our partner company's C and C++ datasets¹, and to an open-source 320 million LOC BigCloneBench (BCB) dataset [6], SSCD was both effective and efficient, outperforming baseline approaches such as SAGA and SourcererCC [5].

¹<https://github.com/SFI-Lero/SSCD>

Extending this work, research has focused on approaches to improve the clone-detection recall (focusing on the underlying AI-based inference component), while also maintaining precision. The rationale for higher recall is to find as many clones as reasonably possible, for example, to assist with vulnerability detection. Improving accuracy of AI-based artificial neural network (ANN) models is a non-trivial task and several strategies can be adopted. For example:

- Designing new ANNs [7], or re-designing and changing their architecture, parameters, or inputs [4]
- Pre-training and fine-tuning ANNs [8]
- Incorporating alternative ANNs
- Ensemble learning or inference: using multiple ANNs, designed and trained towards a similar task, working together [9]

Our literature review suggests that differing ANNs have not yet been assessed for clone detection at scale. Hence, in the work reported here, we focus on the two latter-most strategies: assessing alternative, state-of-the-art ANNs for clone detection and ensemble inference. These strategies were selected because intuitively state-of-the-art models would seem to promise improved accuracy of downstream tasks [9] with a low practical-application barrier, which can be valuable in industrial research. Likewise, ensembles of state-of-the-art ANNs would seem likely to increase efficacy further, at least in terms of recall. Here, four existing transformer-architecture based [7] ANNs were used:

- CodeBERT (fine-tuned for clone detection)² (CBF) [3], [5];
- GraphCodeBERT (GCB) [4];
- CodeT5 (CT5) [10], and
- OpenAI ADA version 2 (ADA) GPT-3 model [11].

These models were selected because of reported state-of-the-art results in source code tasks (including clone detection). Yet these models are distinct enough in their characteristics to suggest possible recall improvement when used in combination, leveraging their individual strengths and capabilities.

To enhance performance, first, averaging, pipelining, and stacking were considered as ensembling strategies. For example, using the stacking, the embedding outputs of individual models were combined to a higher-level “meta-model” embedding. However, the strategy did not yield effective.

Here in our adopted approach, we determine the results for each ANN individually and then combine the results of these models in all possible combinations (ensemble inferencing) towards improving the accuracy of the downstream task: clone detection. This was done to address the following research question (RQ):

- 1) **How do differing ANNs affect the recall and overall efficacy of a large-scale, AI-based, clone-detection approach like SSCD?**
- 2) **How does ensemble inference, based on combining these differing ANNs, affect the recall and overall**

efficacy of a large-scale, AI-based, clone detection approach like SSCD?

The contributions of this work are following:

- It demonstrates that, by using ensemble inference (combining the outputs of the four selected ANNs), recall with respect to clone detection is improved, as compared to if these models were used individually. Additionally, and somewhat surprisingly, the decrease in precision when using these ensembles is not as drastic as one might expect.
- It demonstrates significant difference in recall and precision performance between four different, state-of-the-art source-code-targeted ANNs, when applied individually;
- In doing so, it clearly identifies the ChatGPT ADA model as most appropriate for high-accuracy clone detection;

The rest of the paper is organized as follows: in Section II, ANNs are introduced and the selected ANNs are discussed in detail, with their differences explained. Also the section, briefly discusses SSCD and changes to SSCD that were needed to run the “alternative ANNs” experiment that is the focus of this paper. Section III presents the experimental methodology and Section IV talks to the results of that experiment. Section V touches on threats to the validity of this experiment and Section VI summarizes conclusions, giving directions for future work.

II. BACKGROUND

In the last decade new, deep neural networks with many layers have become the most accurate computer-based solution to a growing number of problems. The first big successes were in the area of image classification and processing, where ANNs give more accurate results than classical computer vision. More recently, deep neural network models have also greatly improved the capacity and accuracy of text and language processing. Recurrent neural networks (RNN), where connections between the nodes in the neural network can form a cycle, and thus allow output from nodes to affect subsequent input to those nodes, have proven especially accurate for text processing, but they require a great deal of time to train.

Transformer-based models (described below), such as Bidirectional Encoder Representations from Transformers (BERT), allow greater parallelism in the training compared to RNNs, meaning that much larger models can be trained. More recently, generative pre-trained transformer (GPT) models with billions of parameters have achieved even higher accuracy in language processing and generation. These models have been trained to operate both on natural language and on programming languages.

A. Transformer based Models Used in This Work

ANNs used in this work, and their characteristics, are shown in Table I. Abbreviations used in the table (and further in the text) are as follows: PT stands for “pre-trained”; where we use the existing model without modification. FT stands for “fine-tuned”, meaning that we have taken the original pre-trained model and done additional fine-tune training using

²<https://huggingface.co/mchochlov/codebert-base-cd-ft>

TABLE I
CHARACTERISTICS OF TRANSFORMER-BASED ANNS USED IN THIS WORK

Model	Training Status	Input	Year	# Parameters	PL used for training	Embedding Size
ADA	N/A	NL/PL	2022	N/A	N/A	1536
CT5	PT	NL/PL (focus on identifiers)	2021	223M	Ruby, JavaScript, Go, Python, Java, PHP, C, C#	768
CBF	PT/FT	NL/PL	2022	125M	PT: Go, Java, JavaScript, PHP, Python, Ruby FT: Java (BCB)	768
GCB	PT	NL/PL/ data flow graph	2020	125M	PT: Go, Java, JavaScript, PHP, Python, Ruby	768

clone detection training data. The input to the model may be “natural language” (NL), or “programming language” (PL). In the “# Parameters” column, M stands for “millions”. OpenAI ADA version 2 (ADA) is a proprietary GPT-3 based model developed by OpenAI [11]. Some of its parameters are unknown and therefore there is a N/A designation in that cell, as with the programming language(s) used for training.

Common to all ANNs used in this work is what’s known as a transformer architecture [7]. Characteristic of this architecture is the usage of attention layer(s) that allow for learning of a word/token representation from its context. The encoder part of these models takes source code as an input and, using the tokens’ contexts, returns an associated numeric representation (embedding). All the models were pre-trained: they learned a certain generic language model making them suitable for a broad spectrum of NL/PL tasks.

Similarly, all of the models are fairly recent (having appeared in the past 3 years) and have showed state-of-the-art results in natural and programming language tasks particularly, suggesting their suitability for the clone detection task probed here [4], [5], [10], [11].

The differences in these models can be found in their degree of training, input details, their number of parameters, the types of programming languages used for training, and the size of generated embeddings (see Table I). For example, in addition to pre-training, CBF was fine-tuned (further trained) towards clone detection specifically (see our previous work for more details [5]). The input to these models is a mix of NL (e.g. comments) and PL, as found in source code. Unlike other models, CT5 focuses on identifiers in the source code and GCB adds structural information (data flow), derived from source code. CT5 has twice as many parameters as CBF/GCB. The set of programming languages used for training the ANNs includes Ruby, Javascript, Go, Python, Java, and PHP, but is individual to each model. In addition to this set, CT5 was also trained on C (one of the languages used in our industrial partner’s codebase) and C#. Finally, the embedding size in ADA is twice as large as the rest of the models.

The nature of these models (usage directed towards NL/PL tasks) suggests their suitability for clone detection in isolation. Yet their differences suggest they are heterogeneous enough

to be also leveraged together to improve the recall of clone detection: in clone-instances where one model fails another can excel, thus making them more effective cumulatively. The reasoning for relying on the complementary strengths of the chosen ANNs specifically are:

- Fine-tuning with respect to clone detection should improve effectiveness of ANNs in downstream tasks [8];
- The inclusion of structural information could improve clone detection [12];
- The number of parameters in ANNs could improve their effectiveness [13].

We also assume that using the same PL for training and inference (e.g. C language in CT5) can be beneficial. For example, in our previous work, we have fine-tuned CBF on Java-language code-clones, but used that model with C/C++ datasets [5].

Finally, the ADA model belongs to either the GPT3 or GPT3.5 model family [11], [14] that recently showed state-of-the-art results in NL/PL tasks and led to the success of ChatGPT (for example [15]). These factors, and specifically the individual characteristics that suggested individual ANN suitability for clone detection, prompted their selection for inclusion in this study.

B. Scalable clone detection

This section provides an overview of the architecture of SSCD, an approach which is primarily motivated by the need for a more efficient and scalable approach to code clone detection, especially within large-scale code repositories. Several ANN based approaches ASTNN [16], CodeBERT [3], and GraphCodeBERT [17] to clone detection have improved the accuracy of detection of T3/T4 type clones but they rely on pairwise comparison of code fragments and thus can not scale to large code bases due to the quadratic complexity of said pairwise comparisons. In light of these challenges, SSCD was developed to generate numerical representations or ‘embeddings’ for each code fragment using artificial neural networks (transformer architecture). By transitioning to this embedding-based approach, SSCD can leverage efficient approximate k-nearest neighbour (k-NN) algorithms for clone detection, instead of performing exhaustive pairwise comparisons, finding

the 'k' nearest neighbours for each embedding in logarithmic time complexity [5].

The underlying models used in this process are fine-tuned versions of pretrained models—CodeBERT and GraphCodeBERT—which are adapted by adding a pooling layer to generate a 768-dimensional vector representation for each code fragment. For the CodeBERT-fine-tuned model, only textual information from the source code is used to generate vectors, whereas the GraphCodeBERT-fine-tuned model also includes structural information to enhance the richness of the representation.

The tool ranks the results based on their cosine similarity, producing a list of similar code fragments for each piece of source code. The generation of this ranked list is controlled by two parameters: a cosine similarity threshold and a 'topN' parameter that determines the number of results returned.

In summary, the SSCD integrates embedding generation, efficient search techniques, and result ranking, thus providing a scalable and effective solution for clone detection in large codebases.

C. Changes to SSCD in This Work

As can be seen in Figure 1, SSCD has 3 major components: parsing, inference (employing ANNs), and search (relying on k approximate nearest neighbour (kANN) for scalability [18]). The input to the approach is source code and the output is a set of nearest neighbour clone candidates. The approach operates at function-level granularity and its implementing tool currently supports Java, C, and C++ languages.

In our previous work, we relied on locally available models such as CBF and GCB for inference [5]. The ADA model, used in this work is available as a cloud service only [11]. Therefore the capability was added to SSCD to either use locally available models (CT5, CBF, GCB) for inference or to employ a cloud based OpenAI service (ADA) (see Figure 1).

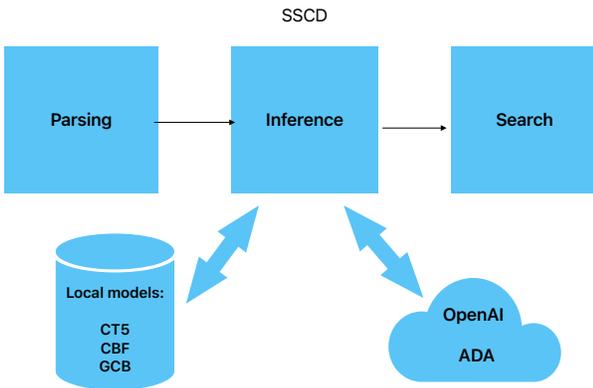


Fig. 1. The Updated Schema of SSCD

SSCD was used in the following manner to evaluate individual-ANN inference and ensemble inference: given a combination of ANNs and source code, SSCD would execute

clone detection with each model selected and capture the results, to assess individual ANN performance. Then, the clone candidates from each individual ANN-run would be merged to assess the combinations. For example, given a combination of CT5 and CBF (CT5_CBF), SSCD would first execute with the CT5 model and then with the CBF model. For the ensemble, the resulting sets of clone candidates from each execution are merged into a final set of clone candidates, removing duplicates (resembling mathematical union operation). We hypothesize that this merging should drive higher recall, in that it aggregates the true-positives from both sets. But also important here is the merging effect on precision: intuitively the false positives in both sets, aggregated together, should negatively impact on precision and the extent of that impact is also of interest in a real-world context. Specifically, false positives increase the effort for developers who typically have to scan all the results of clone-detection techniques, for no clone-detection gain.

III. EXPERIMENTAL DESIGN

The objective of this experiment is to assess the effect of using different individual state-of-the-art ANNs, individually and as components of ensemble inference, towards improving the recall of clone detection over large-scale software systems, while preserving sufficient precision. This leads us to the following research question (RQs):

- 1) **How do differing ANNs affect the recall and overall efficacy of a large-scale, AI-based, clone-detection approach like SSCD?**
- 2) **How does ensemble inference, based on combining these differing ANNs, affect the recall and overall efficacy of a large-scale, AI-based, clone detection approach like SSCD?**

Our industrial partners provided the C and C++ datasets, which we have made available online [19] as resources for other researchers. The existing options for C/C++ benchmark datasets are already quite limited and those that do exist were deemed quite unreflective of code clones in realistic codebases by experts at our industrial partner company. Hence this new dataset was manually created by the company to provide realistic examples that spanned all four clone types.

From these, only clone pairs at function-level granularity are used, resulting in 70 such clone pairs for THE C benchmark and 83 clone pairs for C++ benchmark.

Standard metrics of recall, precision, and F-score [1], [5] are used to assess the effectiveness of the individual ANNs and the subsequent ensemble inferencing, towards clone detection.

To answer the RQs, we assess how using ANN models (listed in Section II), used in isolation and in combination compare to each other, focusing primarily on recall, but also considering precision and overall performance (the F-score). For example, given a combination CT5_CBF, we can assess how its recall compares to CT5 and CBF when the latter two are used standalone, but also how it compares to ADA and GCB. We can also compare it to other ensembles to identify

best-practice for developers. The exact steps to conduct this experiment are as follows:

- 1) First, we run SSCD with both C/C++ datasets using the following parameters: minimum LOC = 0, search = naïve (faiss), code length = 128, no preprocessing (comments etc. retained), (similarity) threshold = 0, and topN = 10 (for parameters and their explanation please see below). In our previous work we found that the code length of 128 allows for effective and efficient neural network inference [5]. The only variable parameter here is the choice of ANN used to compute the embedding for each code segment. We use four ANN models: {ADA, CT5, CBF, GCB} as described in Section II. Therefore, 4 executions of SSCD are conducted: one for each individual model.
- 2) In the second step, we obtain the best performing configuration, with regards to the F-score (given our requirements that improved recall should also have adequate precision). For this, we look at all possible configurations of similarity threshold and topN with given increase-steps, relying on the outputs obtained in the prior 4 executions. The increase-step for the similarity threshold is 0.01 and goes from 0 to 1 inclusively. The increase-step for topN is 1 and goes from 1 to 10 inclusively. Therefore, 100/10 threshold/topN combinations are inspected for each execution (obtained in Step 1). For example, for the ADA model, for the C++ benchmark, the best-performing configuration achieves 96.34% F-score: this happens when the threshold is set to 0.91 and topN is set to 1. This is different for other models.
- 3) Equipped with the results from these best-performing configurations, we combine them in all possible ways, resulting in 11 distinct combinations: {ADA_CT5, ADA_CBF, ADA_GCB, CT5_CBF, CT5_GCB, CBF_GCB, ADA_CT5_CBF, ADA_CT5_GCB, ADA_CBF_GCB, CT5_CBF_GCB, ADA_CT5_CBF_GCB}.
- 4) In the final step, the recall and F-score of the individual ANNs and their combinations are compared to assess their relative efficacy and the effectiveness of the ensembles.

The meaning of the parameters to SSCD are:

- *Minimum LOC*: The threshold number of lines of code that a C/C++ function must contain for SSCD to be considered as a clone candidate.
- *Search*: The efficient nearest neighbour library that is used to find similar embeddings. In this work we use the Faiss library from Facebook.
- *Code length*: The number of tokens of input source code from a given code segment that is used as input to the neural network when creating the embedding. Any subsequent tokens in the code segment are ignored.
- *Pre-processing*: SSCD offers several pre-processing choices to create a canonical form of the source code

for each code fragment, such as rewriting the code to eliminate unnecessary white space. In the current paper we use no pre-processing.

- *Similarity threshold*: The nearest neighbour algorithm computes the cosine similarity between embeddings to find code segments with similar embeddings. Pairs of similar embeddings are the initial set of candidates that SSCD identifies as possible clones. Setting a similarity threshold causes SSCD to eliminate all candidate pairs with a similarity less than the threshold. Note that different neural networks can create neighbours with very different similarity scores. Thus, in the second step above, we search for a suitable similarity threshold for each of the four ANNs.
- *topN*: The nearest neighbour algorithm is configured to find the *topN* nearest neighbours for each embedding. Finding a larger number of nearest neighbours for each embedding increases the execution time. The nearest neighbours for some embeddings may be very close, and for others very distant. Thus, the similarity threshold is used to eliminate neighbours that may be nearest to some embedding, but nonetheless distant.

IV. RESULTS AND DISCUSSION

A. Results

The results of this experiment are presented in Table II and in Table III for C++ and C benchmarks respectively. The upper parts of these tables show recall, precision, and F-score for individual models and the lower parts show these statistics for models' combinations. In Table III, only 4 combinations are used: ADA is excluded because it achieves 100% F-score individually and thus cannot be improved: any combination would only decrease precision.

TABLE II
ENSEMBLE INFERENCE RESULTS FOR C++ BENCHMARK

Model/combination name	Recall (%)	Precision (%)	F-score (%)
Individual models			
ADA	95.18	97.53	96.34
CT5	90.36	96.15	93.17
CBF	81.93	90.67	86.08
GCB	84.34	89.74	86.96
Combinations			
ADA_CT5	98.80	95.35	97.04
ADA_CBF	97.59	91.01	94.19
ADA_GCB	98.80	90.11	94.26
CT5_CBF	92.77	89.53	91.12
CT5_GCB	91.57	88.37	89.94
CBF_GCB	87.95	85.88	86.90
ADA_CT5_CBF	98.80	89.13	93.72
ADA_CT5_GCB	98.80	88.17	93.18
ADA_CBF_GCB	98.80	86.32	92.14
CT5_CBF_GCB	93.98	84.78	89.14
ADA_CT5_CBF_GCB	98.80	84.54	91.12

Overall the results show that ADA is the clear winner in terms of individual efficacy, both in terms of recall and precision. But the recall achieved during clone detection improved

TABLE III
ENSEMBLE INFERENCE RESULTS FOR C BENCHMARK

Model/combination name	Recall (%)	Precision (%)	F-score (%)
Individual models			
ADA	100	100	100
CT5	90	92.65	91.31
CBF	77.14	94.74	85.04
GCB	84.29	90.77	87.41
Combinations			
CT5_CBF	91.43	88.89	90.14
CT5_GCB	91.43	85.33	88.27
CBF_GCB	87.14	87.14	87.14
CT5_CBF_GCB	92.86	82.28	87.25

in all cases (15/15) when ensemble inference is compared to its component elements. For example, the ADA_CT5 combination used on the C++ benchmark achieves 98.8% recall, while, when used individually, the constituent models achieve only 95.18% (ADA) and 90.36% (CT5).

For the ADA_CT5 combination, the precision drops by 2.18% compared to the ADA model and 0.8% compared to the CT5 model. The worst-case precision drop, from the best individual model (ADA, 97.53%) to the four-model combination ADA_CT5_CBF_GCB (84.54%), is a decrease of about 13.33%. When we exclude ADA, the highest precision drop is from the individual CT5 model (96.15%) to the three-model combination CT5_CBF_GCB (84.78%), which is a decrease of approximately 11.82%.

It is also important to note, that this recall (98.8%) cannot be achieved by either ADA or CT5 individually, while maintaining the same precision (95.35%): ADA cannot achieve this recall individually with any threshold/topN parameters in the threshold range of $\{0 \dots 1\}$ and the topN range of $\{1 \dots 10\}$. In contrast, CT5 can achieve this recall (98.8%) individually (the recall was manually increased to this number), but its precision and F-score decrease very markedly to 16.3% and 27.98% respectively.

Looking at the results on the C benchmark, the individual ADA model reports perfect results, achieving maximum recall and precision. No combinations of ADA were considered as they would have only decreased precision for no possible recall gain. The best alternative individual model was CT5 achieving 90% recall and 92.65 precision, but again there was a wide span in performance across the individual techniques.

In terms of ensembles CT5_CBF achieved a relatively high recall (91.43%) with precision of 88.89%. While a higher recall was achieved by CT5_CBF_GCB, that did come at the expense of precision, which decreased to 82.28%.

The answers to the RQs posed above then are as follows:

- 1) Differing ANNs significantly impact the recall and overall efficacy of SSCD-based clone detection, with ADA outperforming the other individual ANNs trialled;
- 2) Ensemble inference improves recall further over the component ANNs (as expected) and, in the case of the ADA-CT5 ensemble, improves overall efficacy, as mea-

sured by the F-score. For other ensembles, while recall persistently improves, overall efficacy often deteriorates due to increased imprecision.

B. Discussion

In terms of the individual ANNs, ADA outperforms the others significantly, particularly with respect to recall and overall F-score. CT5 is quite close in terms of precision but for best clone-detection performance, ADA is the most suitable candidate.

However, ADA is a GPT-based model that is not publicly available. To use ADA we had to send each of our code fragments to the OpenAI platform, and pay a small dollar amount for ADA to generate the embeddings for each code fragment. In many cases, proprietary source code is a valuable business asset and, for such organizations, sending their source code to an external provider is unlikely to be acceptable.

If that is the case, then the best individual option is CT5: recall drops by 5% on the C++ benchmark (10% on the C benchmark) but code confidentiality is preserved. Alternatively the CT5_CBF_GCB ensemble could also be considered: using that approach improves recall to nearly 94% on the C++ benchmark (92.86% on the C benchmark) whilst ensuring confidentiality. Lower precision means the need to look at more false candidates. However, even though precision for this ensemble comes in as second-lowest across all approaches, a precision rate of 82-84% does imply that 16-17 out of every 20 candidates proposed by the approach would be true positives, so not too much developer effort would be wasted during confirmation. A final point that needs to be made about this alternative is that all three approaches can be run in parallel (hardware resources permitting) meaning that speed performance is limited only by the slowest performing individual technique.

In terms of best-efficacy overall, regardless of inference-location, the choice for C is the individual ADA approach, albeit based on this admittedly small dataset. For C++ the choice is slightly less clear: ADA offers the best precision but recall and overall F-score is achieved using the ADA_CT5 ensemble.

V. THREATS TO VALIDITY

We acknowledge ANN selection and sampling bias as one of the core threats to the validity of this experiment. Although, we've tried to select transformer-based ANNs with distinctive characteristics, those achieving state-of-the-art results, and those produced by different research teams (OpenAI - ADA; Microsoft - CBF, GCB; CT5- Salesforce), it is still possible that there are other existing models of this architecture-family (transformers) that would achieve better results and affect the ensemble results of this experiment by providing more diverse individual characteristics.

In addition, the sample size of of the code benchmarks is fairly small: it was approximately 480 KLOC in total with 153 clone, and was exclusively composed of C/C++ code. This suggests that the initial findings presented here should

be buttressed by larger-scale studies and studies of different languages.

But good large-scale datasets are exceptionally difficult to locate due to the nature of clone detection [20]. For example, while the code employed here was scanned for additional clones, it is unlikely that the 153 clones identified as our gold-standard reflect all the clones in the dataset. Consequently our results may reflect a slightly higher recall than is absolutely correct and a slightly lower precision. To mitigate against this latter concern we manually inspected all 'false positives' generated by the ANNs and found that all were indeed correctly categorized.

VI. CONCLUSIONS AND FUTURE WORK

In this work we employ differing state-of-the-art ANNs and ensembles of those ANNs to assess their effectiveness towards improving the recall of clone detection, with a view to improving large-scale clone detection. The results (see Section IV) suggest that ADA is the best individual technique and that typically ensemble inference can be used to improve the recall: it improved over the individual components in 15/15 cases assessed, although combinations of ADA were not considered for the C dataset because it had already achieved 100% recall and precision in isolation. In this context, it's worth mentioning that the C dataset is relatively small, with just 80,190 lines of code (LOC), in contrast to the significantly larger C++ dataset, which contains 424,626 LOC. Thus the C++ dataset might have allowed for a more realistic evaluation.

There are several practical implications here:

- In situations where sending source code to the cloud is acceptable, ADA seems like a promising alternative;
- Where ADA is not acceptable, CT5 is the best individual candidate;
- Ensemble inference can improve the recall of clone detection, if needed, and seems to have a low-cost practical application barrier;
- Ensemble inference can be helpful particularly if privacy and/or cost considerations exist. For example, the CT5_CBF_GCB combination achieves nearly as good recall (93.98%) as ADA (a proprietary, paid-for service) (95.18%) on the C++ dataset (see Table II). On the C dataset there is a larger distance between the approaches but the free, non-proprietary ensemble still identifies 92.86% of the clones with acceptable precision.
- Inference time was not specifically assessed here but it depends on the execution design. When executed in parallel, the inference time becomes the maximum time of any ANNs in the ensemble. When executed sequentially, the inference time is a sum of all inference times of all the ANNs in the ensemble.

Future work might include characterization of other ANN models and inspecting the impact of their characteristics towards improving the recall of clone detection (whilst preserving precision). Another direction might use larger benchmarks, such as the BCB [6], while remaining cognisant of the limitations of these larger datasets [20].

ACKNOWLEDGMENT

This work was supported, in part, by Science Foundation Ireland grant 16/RC/3918 and by Huawei Technologies Co., Ltd.

REFERENCES

- [1] D. Rattan, R. Bhatia, and M. Singh, "Software clone detection: A systematic review," *Information and Software Technology*, vol. 55, pp. 1165–1199, 2013.
- [2] Q. U. Ain, W. H. Butt, M. W. Anwar, F. Azam, and B. Maqbool, "A systematic review on code clone detection," *IEEE Access*, vol. 7, pp. 86 121–86 144, 2019.
- [3] Z. Feng, D. Guo, D. Tang, N. Duan, X. Feng, M. Gong, L. Shou, B. Qin, T. Liu, D. Jiang, and M. Zhou, "Codebert: A pre-trained model for programming and natural languages," 2 2020. [Online]. Available: <http://arxiv.org/abs/2002.08155>
- [4] D. Guo, S. Ren, S. Lu, Z. Feng, D. Tang, S. Liu, L. Zhou, N. Duan, A. Svyatkovskiy, S. Fu, M. Tufano, S. K. Deng, C. Clement, D. Drain, N. Sundaresan, J. Yin, D. Jiang, and M. Zhou, "Graphcodebert: Pre-training code representations with data flow," 9 2020. [Online]. Available: <http://arxiv.org/abs/2009.08366>
- [5] M. Chochlov, G. A. Ahmed, J. V. Patten, G. Lu, W. Hou, D. Gregg, and J. Buckley, "Using a nearest-neighbor, bert-based approach for scalable clone detection," in *2022 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, 2022, pp. 582–591.
- [6] J. Svajlenko and C. K. Roy, "Bigcloneeval: A clone detection tool evaluation framework with bigclonebench," 2016, pp. 596–600.
- [7] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, "Attention is all you need," *Advances in neural information processing systems*, vol. 30, 2017.
- [8] J. Devlin, M. W. Chang, K. Lee, and K. Toutanova, "Bert: Pre-training of deep bidirectional transformers for language understanding," *arXiv preprint arXiv:1810.04805*, 2018.
- [9] S. Ghosh and A. Chopra, "Using transformer based ensemble learning to classify scientific articles," in *Trends and Applications in Knowledge Discovery and Data Mining: PAKDD 2021 Workshops, WSPA, MLMEIN, SDPRA, DARAI, and AI4EPT, Delhi, India, May 11, 2021 Proceedings 25*. Springer, 2021, pp. 106–113.
- [10] Y. Wang, W. Wang, S. Joty, and S. C. H. Hoi, "Codet5: Identifier-aware unified pre-trained encoder-decoder models for code understanding and generation," 9 2021. [Online]. Available: <http://arxiv.org/abs/2109.00859>
- [11] "New and improved embedding model," <https://openai.com/blog/new-and-improved-embedding-model>, 2023, [Online; accessed 09-June-2023].
- [12] L. Jiang, G. Mishergahi, Z. Su, and S. Glondu, "Deckard: Scalable and accurate tree-based detection of code clones," 2007, pp. 96–105.
- [13] A. Neelakantan, T. Xu, R. Puri, A. Radford, J. M. Han, J. Tworek, Q. Yuan, N. Tezak, J. W. Kim, C. Hallacy *et al.*, "Text and code embeddings by contrastive pre-training," *arXiv preprint arXiv:2201.10005*, 2022.
- [14] T. Brown, B. Mann, N. Ryder, M. Subbiah, J. D. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell *et al.*, "Language models are few-shot learners," *Advances in neural information processing systems*, vol. 33, pp. 1877–1901, 2020.
- [15] E. Kasneci, K. Seßler, S. Küchemann, M. Bannert, D. Dementieva, F. Fischer, U. Gasser, G. Groh, S. Günemann, E. Hüllermeier *et al.*, "Chatgpt for good? on opportunities and challenges of large language models for education," *Learning and Individual Differences*, vol. 103, p. 102274, 2023.
- [16] J. Zhang, X. Wang, H. Zhang, H. Sun, K. Wang, and X. Liu, "A novel neural source code representation based on abstract syntax tree," in *2019 IEEE/ACM 41st International Conference on Software Engineering (ICSE)*. IEEE, 2019, pp. 783–794.
- [17] D. Guo, S. Ren, S. Lu, Z. Feng, D. Tang, S. Liu, L. Zhou, N. Duan, A. Svyatkovskiy, S. Fu *et al.*, "Graphcodebert: Pre-training code representations with data flow," *arXiv preprint arXiv:2009.08366*, 2020.
- [18] Y. A. Malkov and D. A. Yashunin, "Efficient and robust approximate nearest neighbor search using hierarchical navigable small world graphs," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 42, pp. 824–836, 2018.
- [19] [Online]. Available: <https://github.com/SFI-Lero/SSCD/tree/main/dataset-01>

- [20] J. Krinke and C. Ragkhitwetsagul, "Bigclonebench considered harmful for machine learning," in *2022 IEEE 16th International Workshop on Software Clones (IWSC)*. IEEE, 2022, pp. 1-7.