TCNAS: Transformer Architecture Evolving in Code Clone Detection

Hongyan Xu[2,3], Xiaohuan Pei[4], Xiu Su[1,4], Shan You[5], Chang Xu[4]

[1] Big Data Institute, Central South University

[2] School of Computer Science, Faculty of Engineering, the University of New South Wales, Australia

[3]Data61, The Commonwealth Scientific and Industrial Research Organisation (CSIRO)

[4]School of Computer Science, Faculty of Engineering, The University of Sydney, Australia

[5]SenseTime Research

Motivation

Code clone detection: Code clone detection aims at finding code fragments with syntactic or semantic similarity.

Syntactic and Semantic Similarity Detection: Aiming to find code fragments that are similar either syntactically or semantically.

Semantic Long-Term Context: Most existing methods ignore the alignment of semantic long-term context, focusing primarily on syntactic similarity.

Manual Model Design: Reliance on human-designed models for source code encoding, requiring expert input and extensive time for experimentation and refinement.

Issues:

- The prevailing emphasis on syntactic detection fails to capture the full spectrum of code similarities, missing crucial semantic relationships.
- The dependency on expert-driven manual model development is not only inefficient but also introduces bottlenecks in adapting to new languages or frameworks.
- Existing methods often struggle to generalize across different coding languages and styles, reducing their applicability in varied development environments.
- The challenge of dealing with sparse and noisy data in large codebases, which complicates the task of accurate clone detection.

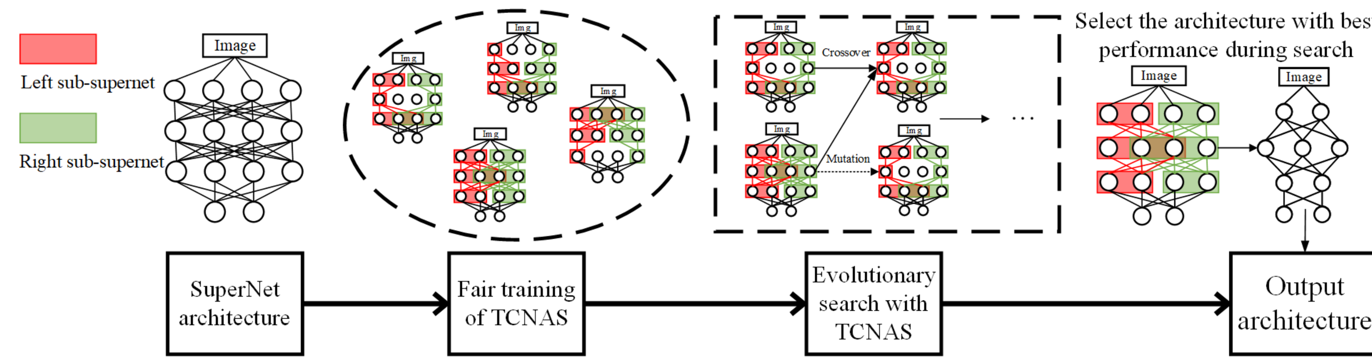
Contribution

- **Innovative Framework Integration:** Proposes a neural architecture search (NAS)-based framework incorporating transformers for code clone detection, effectively capturing long-term dependencies and optimizing model structures.
- **Transformative Code Parsing and Analysis:** Transforms code structure into sequential dataflow, significantly enhancing the extraction of key features by leveraging transformer architecture.
- **Empirical Validation and Superior Performance:** We conduct various empirical experiments on the benchmark, which covering all four types of code clone detection. The results demonstrate our approach consistently yields competitive detection scores across a range of evaluations.

Experimental Settings

- Datasets used: BigCloneBench, which is a standard benchmark for code clone detection. It is abstracted from 60 thousand code snippets which are tagged across ten distinct functionalities. The functionalities of code snippets in the benchmark are widely used the programming scenarios, such as web download and SQL rollback.
- Hardware platform: NVIDIA Tesla V100 GPUs.
- Software platform: Pytorch.
- Setting: Trained for 200 epochs with a batch size of 256, using the AdamW optimiser with a dynamic learning rate reduction strategy.

Framework of proposed model



Specifically, our framework first parses the structure of the code, then transforms it into sequential data of the dataflow. This transformation facilitates key feature extraction by the transformer model.

Main method of TCNAS

Current methods mainly follow a left assigned NAS (LANAS) principle for the evaluation of each width, which induces the training unfairness of channels in supernet.

The probability of training the i-th channel is expressed as $P_i = \frac{n-i+1}{n}$. This makes channels closer to the left train more often, causing uneven training and evaluation bias. This bias affects the supernet's ability to accurately rank performance.

To address this, we introduce a new supernet. It evaluates each width using sub-networks for both left and right channels. Essentially, it consists of two identical networks, S_l and S_r, evaluated using the LA principle but counting channels in reverse. This promotes channel fairness during training. As a result, for each training step:

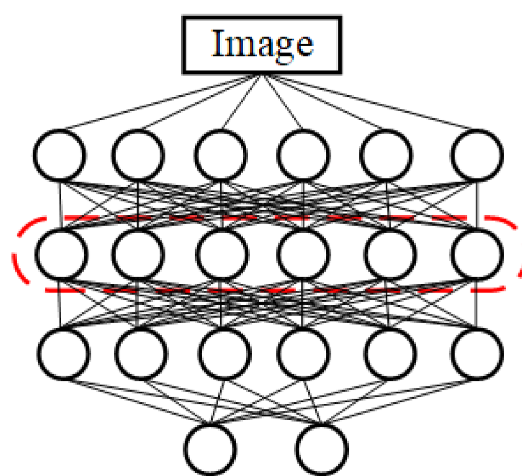
$$\mathcal{T}(d,l) = \mathcal{L}_t(w_d^l; \mathcal{S}, \mathcal{W}, d, \mathcal{D}_{tr}) \quad (1)$$

$$\mathcal{T}(d,r) = \mathcal{L}_t(w_d^r; \mathcal{S}, \mathcal{W}, d, \mathcal{D}_{tr}) \quad \text{s.t. } d \in U(\mathcal{A}), \quad (2)$$

where $\mathcal{T}(d,l)$ and $\mathcal{T}(d,l)$ indicate left and right part of super-network, respectively. With this setting, our super-network is optimized as:

$$\mathcal{W}_{TC}^* = \underset{w_d \in \mathcal{W}}{\arg\min} \underset{d \in U(\mathcal{A})}{\mathbb{E}} [\![\mathcal{T}(d,l) + \mathcal{T}(d,r)]\!] \quad (3)$$

Schematic diagram of TCNAS



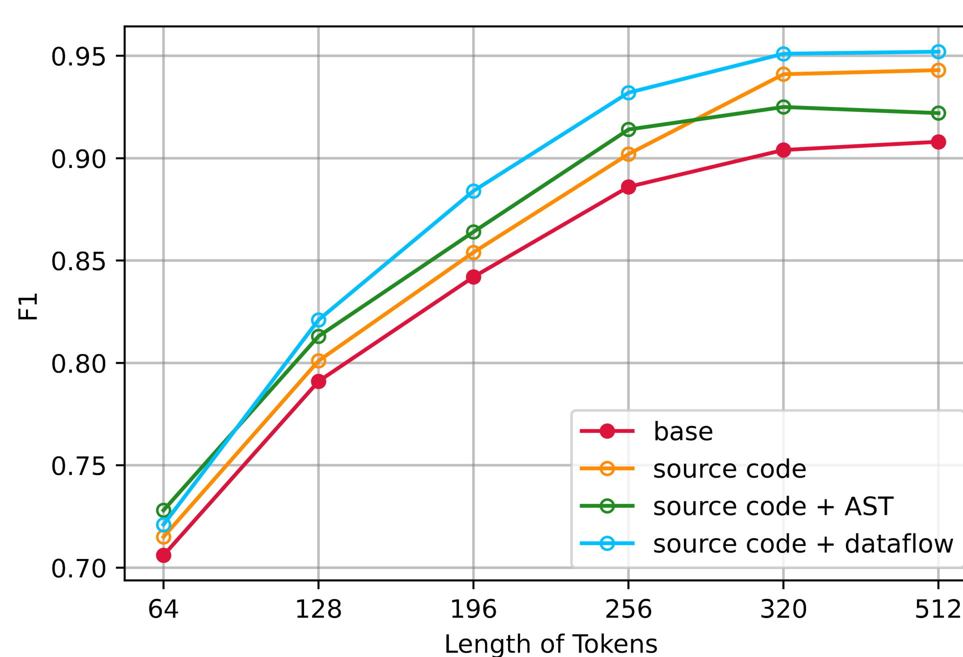
(a)Supernet example

Channels c	$c=1$	$c=2$	$c=3$	$c=4$	$c=5$	$c=6$	Total usage
LaNAS							[6,5,4,3,2,1]
Usage times	[1,0,0,0,0,0]	[1,1,0,0,0,0]	[1,1,1,0,0,0]	[1,1,1,1,0,0]	[1,1,1,1,1,0]	[1,1,1,1,1,1]	
TCNAS							[7,7,7,7,7,7]
Usage times	[1,0,0,0,0,1]	[1,1,0,0,1,1]	[1,1,1,1,1,1]	[1,1,2,2,1,1]	[1,2,2,2,2,1]	[2,2,2,2,2,2]	

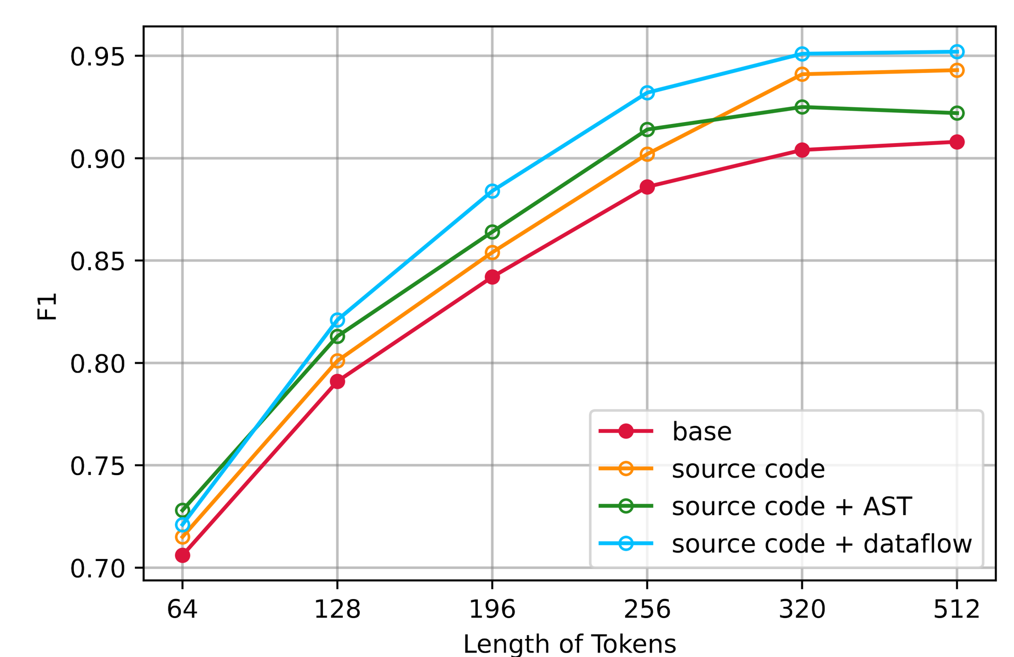
(b)Channel assignment of supernets

Figure (a) A toy example of the supernet. We present an example of a search with 6 searchable channels in each layer. (b) Examples of Different Weight Sharing Patterns. In LaNAS, the 'c' leftmost channels are assigned to the sub-network of width 'c', with all channels being utilized at different times. In contrast, our TCNAS allows all channels to be used an equal number of times, thereby enhancing the supernet's ability to fairly rank all architectures.

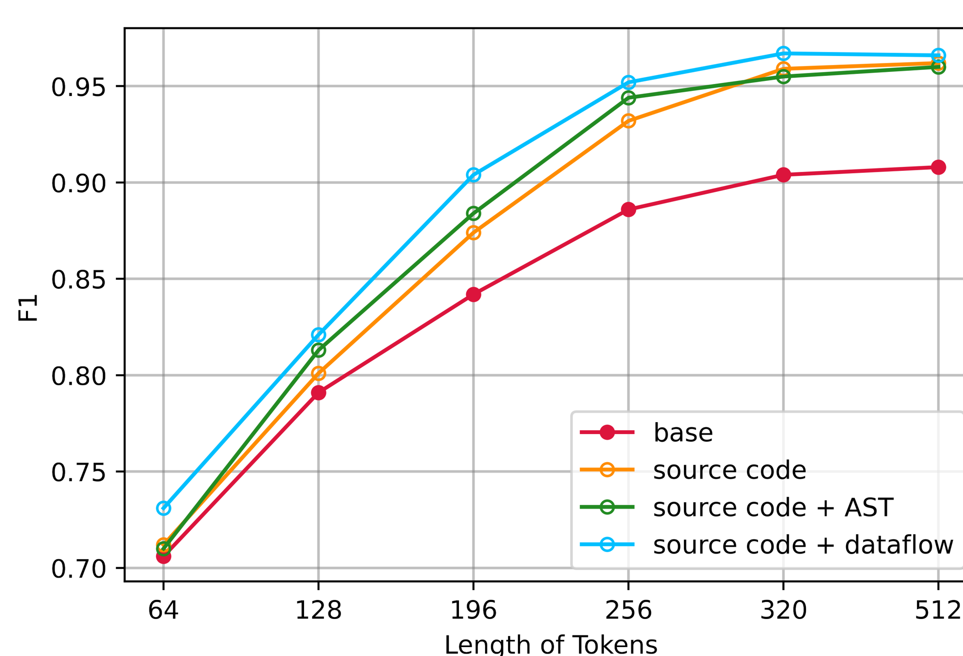
Experimental results on skin cancer dataset



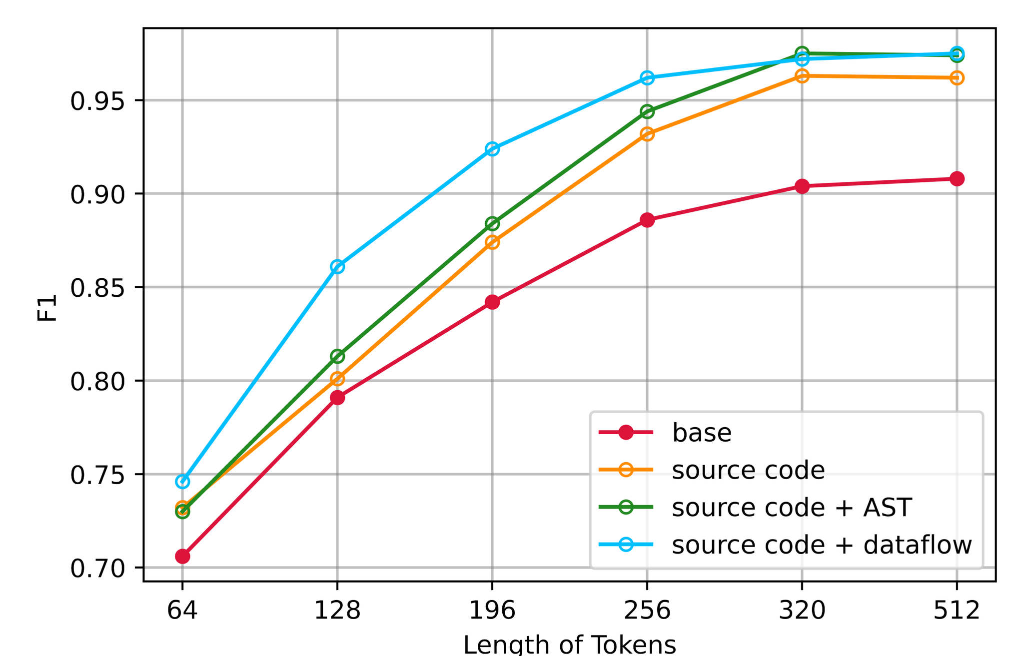
(a) DeiT-small



(b) DeiT-based



(c) Twins-small



(d) Twins-based

Figure: Impact of token's length for each search strategy. Figure shows that using a token type combining source code and dataflow improves model performance. The results reveals dataflow captures code semantics better than traditional syntactical features like tokens and AST nodes.