Improving AST-Level Code Completion with Graph Retrieval and Multi-Field Attention

Yu Xia
xiayu1999@csu.edu.cn
Central South University
Changsha, China

Tian Liang
leungtien@csu.edu.cn
Central South University
Changsha, China

Weihuan Min
csu_sewhm@csu.edu.cn
Central South University
Changsha, China

Li Kuang*
kuangli@csu.edu.cn
Central South University
Changsha, China

ABSTRACT

Code completion, which provides code suggestions by generating code snippets or structures, has become an essential feature of integrated development environments (IDEs). Recently, some studies have begun to use graph neural networks to complete AST-level code, and shown that it is promising to introduce GNNs into AST-level completion. However, these methods do not fully exploit the potential of reference codes with similar structures nor solve out-of-vocabulary (OOV). We propose Retrieval-Assisted Graph Code Completion (ReGCC) to enhance AST-level code completion further. ReGCC integrates a retrieval model that searches for similar code graphs to generate graph nodes and a completion model that leverages information from multiple domains. The key component of both the retrieval and completion models is the Multi-field Graph Attention Block, which consists of three layers of stacked attention: (1) Neighborhood Attention: preserves the heterogeneity and local dependency of the graph, enabling nodes to exchange information within their neighborhood. (2) Global & Memory Attention: addresses the long-distance dependency problem by providing nodes with a global view and the ability to extract information from the memory domain. (3) Reference Attention: lets nodes obtain valuable information from structurally similar reference code graphs. Furthermore, we tackle the OOV issue by employing feature matching and copying values from existing nodes. Specifically, we predict edges between nodes beyond the vocabulary, enabling effective information transfer. Experimental results demonstrate the superiority of our approach over state-of-the-art AST-level completion methods and generative language models.

KEYWORDS

Code Completion, Graph Retrieval, Graph Neural Network

*Li Kuang is the corresponding author.

ACM Reference Format:
Yu Xia, Tian Liang, Weihuan Min, and Li Kuang. 2024. Improving AST-Level Code Completion with Graph Retrieval and Multi-Field Attention. In *32nd IEEE/ACM International Conference on Program Comprehension (ICPC '24), April 15–16, 2024, Lisbon, Portugal.* ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3643916.3644420

1 INTRODUCTION

Code completion is a crucial feature in modern Integrated Development Environments (IDEs), providing developers with code suggestions. The current methods of code completion can be divided into two kinds: Token-level code completion and Abstract Syntax Tree (AST) level code completion.

Token-level code completion sequentially generates code tokens [7, 8, 11, 14, 17, 21, 22, 25, 26, 29, 31, 36]. Current methodologies for code completion, such as Codex[6] and Copilot [10], which are based on Large Language Models (LLMs), can also be classified as Token-level methods since they utilize or enhance text generation techniques. However, it is important to note that Token-level modeling does not strictly adhere to code structures. As a result, the generated code may contain syntactical errors [3].

The AST-level code completion [16, 18, 22, 24, 38, 40] involves a sequential prediction process where the *value* and *type* of nodes are predicted to generate a complete Abstract Syntax Tree (AST), as depicted in Figure 1. This approach predicts the next node based on the previously predicted ones, ensuring the resulting AST corresponds to a syntactically correct piece of code. Additionally, the generated AST are more compatible with the IDE's completion [33]. Therefore, we focus on AST-level code completion in this paper.

Existing AST-level completion methods can be broadly classified into three categories: language model-based (LM-based), path-based, and graph neural network-based (GNN-based). LM-based methods use language models to predict nodes in a flattened AST sequentially. However, the performance of LM-based AST-level completion is unsatisfactory for two reasons: on the one hand, serializing AST may lose important topological structure [12, 16, 38]. On the other hand, node prediction struggles to benefit from large text corpora since structured AST is markedly different from natural text or code [43]. Path-based methods explicitly encode paths to express the hierarchical structure of AST and incorporate path information into the completion process [16, 22, 24]. However, the path encoding is decoupled from the global representation of nodes, hindering the complete integration of node information into the paths.

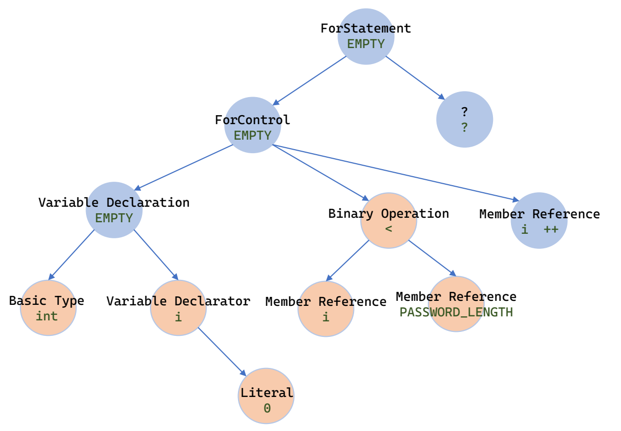


Figure 1: An example of AST-level code completion. According to the preorder traversal, the next node (marked with *?*) is predicted based on all previous ones.

GNN-based methods employ GNNs to perform AST completion [38, 40]. GNNs can preserve structural features in non-Euclidean data by implicitly encoding paths through neighborhood propagation, which is more effective for modeling ASTs. The latest methods, such as CCAG [38] and CC-GGNN [40], have demonstrated the effectiveness of structure modeling in AST completion using GNNs. However, GNN-based AST completion methods still have significant potential for advancement:

First, these methods do not fully exploit the potential of reference codes with similar structures. Empirical studies [1, 27] demonstrate that developers often rely on copying or finetuning existing code fragments for reuse. Recent token-level completion methods have started to consider the behavior, so they retrieve similar code snippets from the corpus and use them as references for the target snippets [26, 42]. However, research on AST-level completion has not considered retrieval as an assist. Some researches [41, 45] also corroborates the viability of structural retrieval: even after undergoing non-Euclidean modeling, the structures of codes with identical functionality continue to demonstrate a high degree of similarity. Therefore, it is promising to retrieve similar ASTs from a large corpus to assist in generating a target AST.

Second, the challenge of OOV problems is overlooked in the existing GNN-based AST completion methods. When the value is absent from the vocabulary, the model cannot provide accurate predictions, ultimately limiting the accuracy. Li et al. [18] try to solve OOV problems with Pointer Networks in language models. However, there is no solution to OOV problems specifically tailored for GNNs, since if Pointer Network is employed directly by inputting the flattened graph, the original edge information in GNNs will be ignored, but the edge relationship is helpful for the reconstruction of OOV nodes. We can try to convert the pointer prediction into edge prediction in GNNs to address the OOV problem.

To address the issues mentioned, we introduce a GNN-based AST-level code completion model, namely the **Re**trieval-assisted **G**raph **C**ode **C**ompletion Model (ReGCC).

As Fig.2 shows, ReGCC is built based on heterogeneous code graphs and consists of a retrieval model and a completion model. The retrieval model aims to identify the most similar ASTs for the target from the corpus so that they can serve as references in the completion model. The completion model generates the AST propagating information through all nodes. Both the retrieval and completion models incorporate a crucial module called Multi-field Graph Attention Block (MGAB), which embodies the effectiveness and innovation of ReGCC.

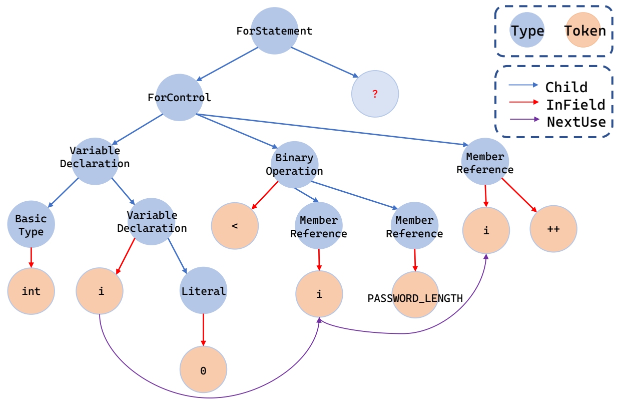


Figure 2: An example of a heterogeneous code graph corresponding to the homogeneous code graph in Fig. 1.

Figure 3 depicts the MGAB for ReGCC. It is designed to gather information from various fields using three forms of attention: neighborhood attention, global & memory attention, and reference attention. (1) The neighborhood attention, realized by Heterogeneous Graph Transformer (HGT) [13], enables nodes to share information with their neighboring peers, thereby preserving their structural relationships; (2) The global & memory attention, realized by self-attention with memory mechanism [9], enables nodes to obtain information from memory and extend their view beyond the subgraph. The attention is particularly effective in handling long-distance dependencies; (3) The reference attention, realized by cross-attention, allows nodes to extract essential information from similar structures. Our design allows nodes to selectively incorporate information from the reference code graph, expanding their vision scope to the reference field.

The retrieval model and completion model of ReGCC both utilize MGAB as their core module, but they have slight differences in implementation. In the retrieval model, two-layers MGABs are employed, consisting of only global & memory attention and neighborhood attention, and a MaxPooling operation for nodes is applied subsequently. In the completion model, MGABs with complete three-layer attentions are stacked.

In addition, to mitigate the OOV problem in the completion model, we propose edge prediction in GNNs as the solution. When the next node has been determined as UNK, the hidden state of the target node will be matched with the hidden states of all previous UNK nodes, aiming to establish whether a NextUse edge connection exists between them; subsequently, we select the most probable match from the set of candidates and copy its value, using it as a predictive value for the new node.

The codes of ReGCC are publicly available on GitHub. [1]

The contributions of this work are as follows:

- Introducing graph retrieval to assist AST generation by mining similar structures. This is the first attempt to employ graph retrieval to assist in code completion.
- Developing an AST MGAB with three layers of attention allows nodes to fully obtain information from the neighborhood, global, memory, and reference fields.
- Proposing a novel solution to mitigating the OOV problem in AST completion. The solution restores the dependencies of UNK nodes by performing node feature matching and edge prediction.
- The experimental results indicate that our method outperforms the state-of-the-art AST completion methods and the latest pre-trained language models regarding accuracy for both *value* and *type.*

2 RELATED WORKS

Current code completion approaches can be divided into two categories: Token-level completion and AST-level completion.

2.1 Token-level Code Completion

Token-level completion approaches treat code as a sequence of tokens and aim to predict the missing tokens at the target location. Various neural network-based models have been proposed for token-level code completion. Svyatkovskiy et al. [32] developed a general framework for encoding serialized source code differently. Liu et al. [25] proposed a multi-task learning-based pre-trained language model for code completion. Izadi et al. [14] utilized type sequences of source code as auxiliary information for training multiple GPT-2 models. Ciniselli et al. [7, 8] explored multi-token completion on the Roberta and T5 models by directly inputting the source code as sequences. Non-autoregressive models have also been proposed to solve single-line completion tasks [21]. Additionally, code retrieval has been integrated into token-level completion models to leverage similar codes for code completion [26]. Guo et al. [11] proposed sketching the code first and leaving the hard-to-predict tokens for developers.

In summary, token-level code completion views the code as a sequence of tokens and fills in the missing tokens. The effectiveness of these models in addressing code completion has been demonstrated by encoding the serialized source code in various manners. However, due to the absence of strict rule constraints, token-level completion cannot guarantee the generation of syntax error-free code in every instance.

Indeed, the issues encountered by token-level completion differ significantly from those faced by AST-level completion. Therefore, in this paper, we only mention token-level completion strategies but **do not include them in the baselines**.

[1]https://github.com/luoshui6/ReGCC

2.2 AST-level Code Completion

AST-level completion occurs in the AST tree, continuously completing the next AST node based on the assumption that the complete code structure is known. Li et al. [18] introduced attention and pointer networks to solve the OOV problem. Liu et al. [23] proposed a multi-task learning architecture that uses the path from the prediction node to the root node as an auxiliary task. Kim et al. [16] compared the performance in code completion by entering only the original token sequence and by including the AST root path and the AST DFS sequence as additional inputs. Wang et al. [37] converted AST into generating action sequences to assist code generation on multi-token single-line completion. Wang et al. [38] constructed AST graphs containing node-node edges and parent-child edges and then put them into GAT [35], achieving better effects on the task of single-token code completion. Liu et al. [24] propose a general framework that can cope with AST-level and Token-level completion. Yang et al. [40] propose CC-GGNN, which can construct AST graphs based on AST trees and combines code semantic and structural information to complete code completion.

Recent AST-level completion research has shifted from language models to graph neural networks, and structured modeling approaches are more suitable for AST completion. However, current GNNs methods have not addressed issues such as fully utilizing retrieval references, OOV, and long-distance dependencies, which require new solutions in GNNs.

3 APPROACH

3.1 Preliminaries

AST-level code completion generates AST nodes sequentially. At each period, the model predicts the *type* and *value* of the next node based on the previous nodes in the pre-order traversal, as Fig. 1 shows.

Existing code completion approaches [18, 23, 24, 38, 40] typically model ASTs as homogeneous trees or graphs. They represent nodes by combining the representation of the node's *type* N_t and its *value* N_v. For non-leaf nodes, N_v is set to a special value *EMPTY.* In this case, the target model f to be learned can be formally defined as shown in Eq. 1.

$$\begin{aligned} N^i &= N_t^i || N_v^i \\ S^n &= [N^0, N^1, \ldots, N^n] \\ N^{i+1} &= f(S^i, E^i) \end{aligned} \tag{1}$$

where || represents vector fusion operations such as *concat* or *sum.* S^n is a list of nodes of length n arranged in pre-order traversal order. E^i represents the edge relationships corresponding to the nodes in S^i.

There are some drawbacks to homogeneous modeling that encodes *type* and *value* jointly. Firstly, fusing the two representations would confuse the model. Models are expected to exhibit biases when predicting *type* or *value*, where *type* predictions heavily rely on structural information provided by other *type*s, while *value* predictions are influenced by semantic information. In homogeneous models, the model must learn how to dissociate them from the node representation. Secondly, homogeneous graphs might include excessively invalid information. To maintain consistent node distributions, traditional methods encode the value of non-leaf nodes in

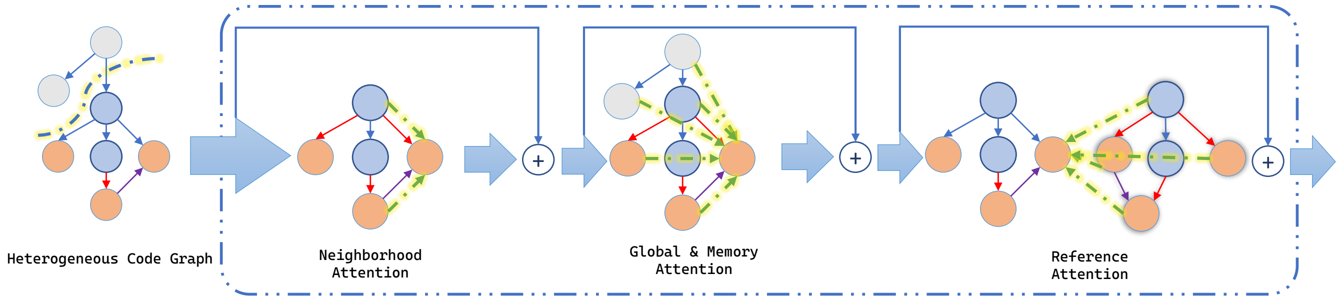


Figure 3: AST MGAB with triple attention.

the AST as a specific marker *EMPTY*. This results in many *EMPTY* with limited meaning. Therefore, we employ heterogeneous graphs instead of homogeneous graphs.

We introduce a heterogeneous code graph to represent ASTs better, as shown in Fig. 2. This graph corresponds to homogeneous modeling in Fig. 1 and separates *type* and *value* into *type* nodes and *token* nodes, respectively. Node relationships are mapped via various edges. The *Child* edge represents the parent-child relationship in the AST tree between *types*. The *InField* edges link leaf nodes' *types* to their corresponding *tokens*. *NextToken* edges connect *tokens* in the order of their occurrence, ensuring the sequential correlation between *tokens*. The *NextUse* edges draw from a *token* to its next appearance, aiding in contextual variable or value identification. In addition, *NextUse* edges enable out-of-vocabulary *tokens* to maintain their dependencies, avoiding confusion with other nodes. This makes them distinguishable even when both are resolved to a special token: <UNK>.

In the heterogeneous graph, we redefine the AST completion objective, which differs from those presented in the Eq. 1. While we continue to flatten the heterogeneous code graph following the pre-order traversal order, it is important to note that nodes within the heterogeneous graph could exist within two entirely different spaces - these include the value space V_{type} for *type* and V_{token} for *token*. Consequently, the AST completion task within the heterogeneous graph is redefined as illustrated in Eq. 2.

$$\begin{aligned} N^i &\in V_{type} \cup V_{token} \\ S^n &= [N^0, N^1, \ldots, N^n] \\ N^{i+1} &= f(S^i, E^i) \end{aligned} \tag{2}$$

ReGCC consists of two models: the retrieval model and the completion model. In both models, AST Multi-field Graph Attention Block (MGAB) serves as the key component. In the subsequent sections, we will illustrate the MGAB and present the two models.

3.2 MGAB

Fig. 3 shows the key component of ReGCC: AST Multi-field Graph Attention Block (MGAB). Aiming at gathering information from various fields of each node, MGAB includes three types of attention mechanisms: neighborhood attention, global and memory attention, and reference attention.

3.2.1 *Neighborhood Attention.*

The first layer of MGAB employs a Heterogeneous Graph Transformer (HGT) [13] to capture the edge relations and the heterogeneity among nodes. HGT offers three essential enhancements compared to the self-attention mechanism. Firstly, it allows a node t to compute attention with its neighboring nodes $s \in N(t)$ that are directly linked by an edge. Secondly, nodes of different types are projected onto different vector spaces, while nodes of the same type $\tau(s)$ share the same query projection matrix $Q\text{-Linear}^i_{\tau(s)}$. Projections for key and query also follow a similar pattern. Thirdly, HGT introduces a meta-path scaling factor (defined as $\mu\langle\tau(s), \phi(e), \tau(t)\rangle$) to compute the attention from node s to node t connected by edge e, thereby facilitating adaptive learning of the relevancy of different meta-paths by the model. Neighborhood attention is calculated via HGT as is exhibited in Eq. 3.

To uphold heterogeneity among nodes and to capture edge relations, the first layer of our MGAB employs a Heterogeneous Graph Transformer (HGT) [13]. HGT offers three essential enhancements to the self-attention mechanism. Firstly, it delimits the field of view of a node, thus allowing a node t to compute attention with its neighboring nodes $s \in N(t)$ that are directly linked by an edge. Secondly, nodes of differing types are distinctly projected onto varied vector spaces, where nodes of the identical type $\tau(s)$ share the same query projection matrix $Q\text{-Linear}^i_{\tau(s)}$. The same is true for the projection of key and value. Thirdly, HGT introduces a meta-path scaling factor $\mu\langle\tau(s), \phi(e), \tau(t)\rangle$ to compute the attention from node s to node t connected by edge e, thereby facilitating adaptive learning of the relevancy of different meta-paths by the model. Neighborhood attention is calculated via HGT as is exhibited in Eq. 3.

$$\begin{aligned} &\text{Attention}_{HGT}(s,e,t) = \text{Softmax}_{\forall s \in N(t)}(\|_{i\in[1,h]}\, ATT(s,e,t)) \\ &\text{ATT}^i(s,e,t) = (K^i(s) W^{ATT}_{\phi(e)} Q^i(t)^T) \frac{\mu\langle\tau(s),\phi(e),\tau(t)\rangle}{\sqrt{d}} \\ &K^i(s) = K\text{-Linear}^i_{\tau(s)}(H^{(l-1)}[s]) \\ &Q^i(t) = Q\text{-Linear}^i_{\tau(t)}(H^{(l-1)}[t]) \end{aligned} \tag{3}$$

, where ATT is the single-head attention calculation, $W^{ATT}_{\phi}(e)$ is the unique weight of edges with the type of $\phi(e)$, and $\|$ represents the concatenation operation.

The neighborhood attention layer within the AST MGAB implicitly facilitates the encoding of hierarchical relationship among nodes. Separate projection matrices for *type* and *value* nodes make them exist in independent spaces while preserve their distribution. In addition, HGT exhibits the capability to adaptively adjust the weights of different meta-path neighbors, which proves to be highly advantageous in the context of a heterogeneous code graph. Therefore, HGT is a good choice for neighborhood attention.

3.2.2 Global & Memory Attention. Distant nodes within a single subgraph or across two subgraphs also need to communicate. For the former case, it is challenging to preserve communication between distant nodes in GNNs, as only neighboring nodes can exchange information, and enough hops is necessary to guarantee interaction between any two nodes [4]. In the latter scenario, scale issues could lead to the division of the graph. However, this partitioning of subgraphs inevitably results in the loss of dependencies emanating from the preceding subgraph. This loss is particularly noticeable for the initial few nodes of the subsequent subgraph, as they may lack sufficient information to guide their generation.

To address the long-distance dependency problem within a subgraph, each node is given a global view and can see all the nodes before it. We apply the multi-head self-attention proposed by Vaswani et al. [34]. This mechanism ensures nodes can glean information from both proximal and distant nodes equally.

To address the long-distance dependency problem across subgraphs, we take the Transformer-XL [9], which allows several nodes at the end of the previous subgraph G_τ (gray nodes in Fig. 3.) to participate in the global propagation. However, these nodes do not participate in the backpropagation of the gradient. For the subgraph $G_{\tau+1}$, the hidden state $\mathbf{h}^n_{\tau+1}$ of all nodes in the n^{th} AST MGAB in the Global & Memory Attention layer is calculated as shown in Eq. 4.

$$\begin{aligned}
\widetilde{\mathbf{h}}^{n-1}_{\tau+1} &= [\mathrm{SG}(\mathbf{M}^{\mathrm{n}-1}_{\tau}) \parallel \mathbf{h}^{\mathrm{n}-1}_{\tau+1}] \\
\mathbf{q}^{n}_{\tau+1}, \mathbf{k}^{n}_{\tau+1}, \mathbf{v}^{n}_{\tau+1} &= \mathbf{h}^{n-1}_{\tau+1}\mathbf{W}^{\top}_{q}, \widetilde{\mathbf{h}}^{n-1}_{\tau+1}\mathbf{W}^{\top}_{k}, \widetilde{\mathbf{h}}^{n-1}_{\tau+1}\mathbf{W}^{\top}_{v} \\
\widetilde{\mathbf{h}}^{n-1}_{\tau+1} &= \text{Masked-Multi-Attn}(\mathbf{q}^{\mathrm{n}}_{\tau+1}, \mathbf{k}^{\mathrm{n}}_{\tau+1}, \mathbf{v}^{\mathrm{n}}_{\tau+1})
\end{aligned} \tag{4}$$

where SG() is the gradient freezing operation, $\mathbf{M}^{n-1}_{\tau}$ is the hidden state of the last several nodes of the subgraph G_τ at the $n-1$ level, and **Masked-Multi-Attn**() is the one-way visible unidirectional multi-head self-attention mechanism. For clarity of the statement, the multi-head operation is omitted.

In this layer, nodes assimilate information from both global and memory fields. The heterogeneous code graph can more effectively preserve long-distance dependencies by integrating the global attention mechanism with the memory mechanism.

3.2.3 Reference Attention. Developers often accelerate software development by copying or modifying existing code components [27].Notably, its core structure generally remains consistent in both cases. Therefore, for AST completion, similar code structures can provide valuable references. Given a target code graph G_τ that requires completion and a similar reference code graph G_γ, we employ asymmetric cross-attention to allow nodes in G_τ to refer to those in G_γ. For instance, the hidden state $\mathbf{h}^n_\tau$ of the reference attention layer in the n^{th} AST MGAB of graph G_τ is computed as Eq. 5.

$$\begin{aligned}
\mathbf{q}^{n}_{r} &= \mathbf{h}^{n-1}_{\tau}\mathbf{W}^{\top}_{q} \\
\mathbf{k}^{n}_{\gamma}, \mathbf{v}^{n}_{\gamma} &= \mathbf{h}^{n-1}_{\gamma}\mathbf{W}^{\top}_{q} \\
\mathbf{h}^{n}_{\tau} &= \text{Multi-Attn}(\mathbf{q}^{n}_{r}, \mathbf{k}^{n}_{\gamma}, \mathbf{v}^{n}_{\gamma})
\end{aligned} \tag{5}$$

where $\mathbf{h}^{n-1}_{\gamma}$ refers to the output of G_γ in the $n-1^{th}$ AST MGAB after the Global & Memory Attention layer.

Both the target code graph G_τ and the reference code graph G_γ separately engage in Neighborhood Attention and Global & Memory Attention. This encourages nodes within reference graphs to mutually exchange information. Incorporating reference attention enables the completion of heterogeneous code graphs to derive inspiration from similar code structures.

The three attention layers form an AST MGAB, enabling nodes to gather information from neighbor, global, memory and reference fields. Each layer within MGAB is connected using *Residual Connections* and *Layer Normalization*, ensuring stable gradients. After each MGAB, we also involve an additional bottleneck feed-forward process, where the hidden dimension is initially expanded before being reduced, thus integrating the internal information represented by each node.

3.3 Retrieval Model

The retrieval model aims to retrieve ASTs that exhibit similar structures, which can serve as a reference for the target ASTs. The functional flow of the retrieval model is shown as Fig. 4. The model consists of multiple two-layer AST MGABs and a node-level Max-Pooling operation. Each AST MGAB in the retrieval model incorporates neighborhood attention and global & memory attention components. Subsequently, a Max-Pooling layer is applied to extract the abstract features from entire heterogeneous code graph. This architecture enables the retrieval model to align closely with the completion model. As a result, samples that exhibit high similarities in the retrieval model provide valuable references for completion. Furthermore, initializing equivalent layers in the completion model with the AST MGABs from the retrieval model enables the model to harness the benefits of contrastive learning. This initialization process allows samples with greater similarities to draw closer in terms of distance to the initial parameters, thereby improving the model stability.

The retrieval model is trained using a corpus of code transformations or clone detection. The training process incorporates the contrast learning [39], where the training data include several positive examples with similar structures and negative examples sampled from other instances. As depicted in Fig. 4, positive examples include pairs of source code graph representation $\mathbf{H}_t$ and augmented code graph representation $\mathbf{H}_{t'}$, which is obtained by semantic-preserving transformation of the source code graph. In addition, there are two pairs of negative examples: the sampled code graph representation $\mathbf{H}_s$ randomly selected from other instances with $\mathbf{H}_t$ and $\mathbf{H}_{t'}$, respectively. We aim to maximize the similarity between positive examples while minimizing the similarity between negative ones. We use the loss function designed in DGMS [19], shown in Eq. 6.

$$\mathcal{L}(\theta) = \sum_{<t,t',s>\in\mathbb{T}} max(0, \delta - \cos(\mathbf{H}_t, \mathbf{H}_{t'}) + \frac{1}{2}(\cos(\mathbf{H}_t, \mathbf{H}_s) + \cos(\mathbf{H}_{t'}, \mathbf{H}_s))) \tag{6}$$

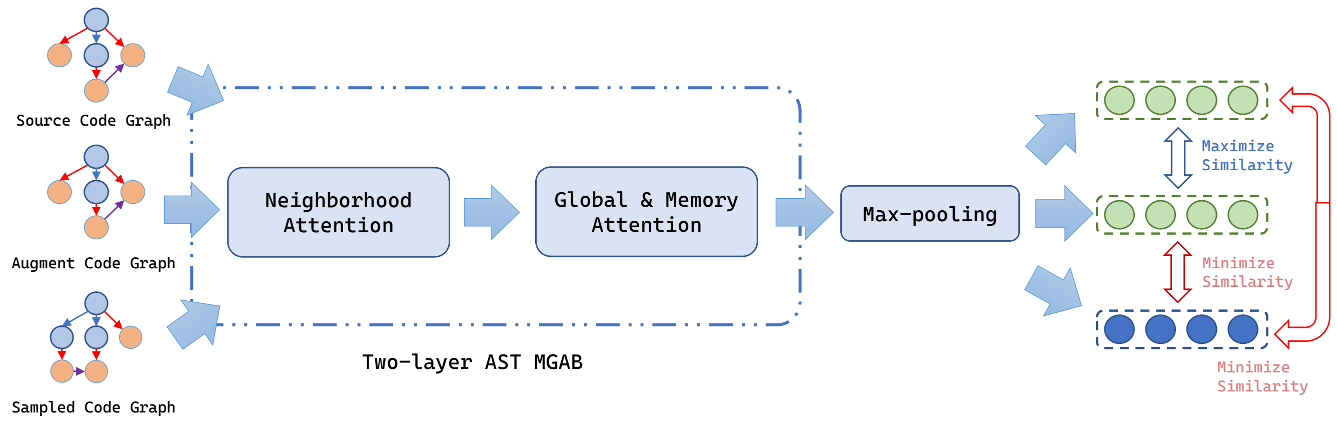


Figure 4: Pipeline of the retrieval model.

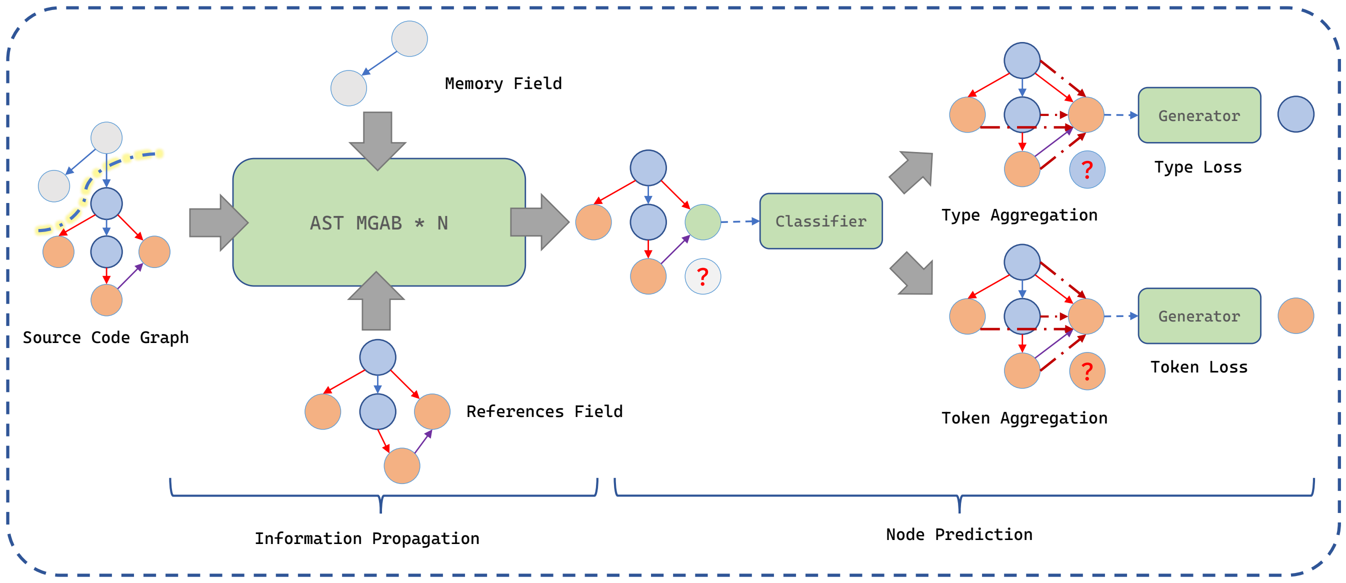


Figure 5: Completion model: stage of information propagation and node prediction.

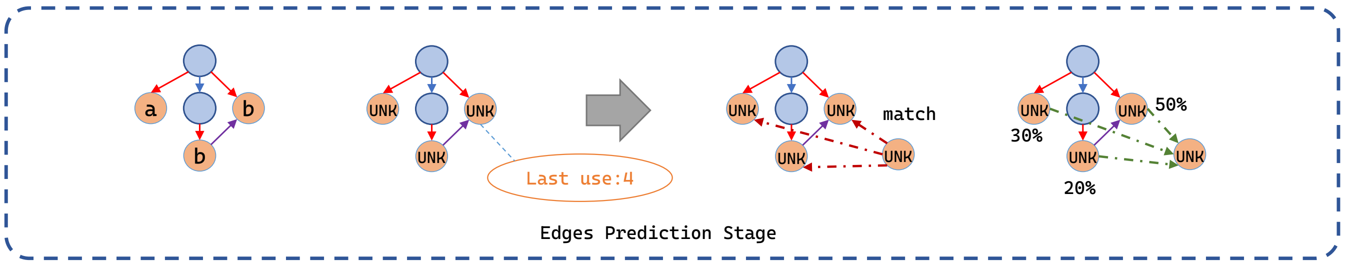


Figure 6: Completion model: stage of edge prediction.

θ denotes the model parameters to be learned and δ is the margin parameter of margin ranking loss.

3.4 Completion Model

The completion model is the central component of ReGCC and follows a three-stage architecture: information propagation, node prediction, and edge prediction. The working flow of the completion model is depicted in Fig. 5.

During information propagation, each node undergoes three layers of propagation via the complete AST MGAB to facilitate information exchange and collection. (Refer to Section 3.2 for details).

In the node prediction stage, we employ the data from the preceding node N_{i-1} to predict the i^{th} target node N_i. Initially, we identify whether N_i is a *type* node or a *token* node, following our redefinition of the complementation scenarios in heterogeneous graphs. To accomplish this, we feed the hidden state H_{i-1} of N_{i-1} into a classifier and acquire the prediction for the node attribute. Subsequently, we append an integration layer specifically designed for attribute prediction following the classifier. We employ the masked self-attention approach to independently maintain distinct parameters for *type*s and *token*s, thereby mirroring their distribution gap. At the final stage, we consolidate the node representation and deliver it to the corresponding generator (*type* or *token* generator) to yield the final prediction.

We designed unidirectional edges and masked self-attention to ensure the one-way flow of information from previous nodes to latter ones in preorder. This design allows us to calculate and predict all nodes in a graph in parallel. In the node prediction, we perform two rounds of masked self-attention at the graph level for both the *type* and *token* aggregations. Based on the node classification result, we extract the hidden state of the target node from either the *type* aggregation graph representation H'_{type} or the *token* aggregation graph representation H'_{token} for the next calculation. We can formally represent the calculation process of the node prediction stage using Eq. 7.

$$\begin{aligned}
p^i_{type}, p^i_{token} &= Softmax(W_c H_{i-1} + b_c) \\
q_{type}, k_{type}, v_{type} &= H\mathbf{W}_q^{\text{Type}}, H\mathbf{W}_k^{\text{Type}}, H\mathbf{W}_v^{\text{Type}} \\
H'_{type} &= \text{ Masked-Multi-Attn } (q_{type}, k_{type}, v_{type}) \\
q_{token}, k_{token}, v_{token} &= H\mathbf{W}_q^{token}, H\mathbf{W}_k^{token}, H\mathbf{W}_v^{token} \\
H'_{token} &= \text{ Masked-Multi-Attn } (q_{token}, k_{token}, v_{token}) \\
T'_{i-1} &= Softmax(W_{gen}^{type} H'_{type}(i-1) + b_{gen}^{type})(p^i_{type} > p^i_{token}) \\
T'_{i-1} &= Softmax(W_{gen}^{type} H'_{token}(i-1) + b_{gen}^{token})(p^i_{type} > p^i_{token})
\end{aligned} \tag{7}$$

where p^i_{type} and p^i_{token} are the probabilities of node i being a *type* node or *token* node, respectively. H_{i-1} is the state of node N_{i-1} before the node prediction stage, and T'_{i-1} is the output of node N_{i-1} in the prediction stage.

The completion model incorporates an additional stage for edge prediction, aimed at addressing the issue of Out-Of-Vocabulary (OOV) tokens. Fig.6 provides an example of edge prediction. Edge prediction specifically focuses on Out-Of-Vocabulary *token* nodes, which are represented as UNK nodes. These nodes cannot yield correct predictions due to the lack of necessary information. During the training phase, we extract all the connection relationships involving UNK nodes to train the model in feature matching. In the testing stage, we execute feature matching between the predicted UNK node and all preceding UNK nodes. The probability of these nodes being connected via a *NextUse* edge is evaluated afterward. The node with highest probability is selected, and its value is predicted as the value pointing to the target UNK node.The feature matching scheme and edge prediction can be referred to Su et al. [28] in the recommendations field. Consequently, we calculate the feature matching between two nodes, N_i^{unk} and N_j^{unk}, as illustrated in Eq. 8.

$$f_{ep}(N_i^{unk}, N_j^{unk}) = W_2^e \mathbf{ReLU}(W_1^e(T'_i \odot T'_{j-1}) + b_1^e) + b_2^e \tag{8}$$

where $\odot$ represents element-wise multiplication. Like the node prediction, the prediction of edges also relies on the hidden state of the previous node N_{j-1} to predict the next node N_j. For edge prediction, we calculate $f_{ep}(N_i^{unk}, N_j^{unk})$ for every N_i^{unk} with $i < j$, and then perform a softmax to obtain the connection probability for each UNK node.

Fig. 6 is an example. Suppose we have a heterogeneous code subgraph with five nodes, numbered according to the preorder traversal. Nodes 2, 4, and 5 represent Out-Of-Vocabulary (OOV) tokens: a, b, b. When the graph undergoes parsing, OOV nodes are all parsed as UNK nodes. Without specific architectural design, our model could not discriminate between the tokens a and b. We maintain this information by implementing the *NextUse* design: Node 2 represents one instance of the token, and nodes 4 and 5 represent another instance of the same token. When predicting a token node *n* after node 5, if this node is determined as UNK by the token generator, we carry out a feature matching process with all preceding UNK nodes, followed by node-level softmax operation. Ultimately, the UNK node with the highest probability is taken as the value for node *n*. During the testing phase, we identify the most probable node within the context of the UNK node that is linked to the target node via a *NextUse* edge and predict the value of the target node to align with that of the identified node.

In summary, the completion model optimizes four parts of the loss simultaneously: the loss of the attribute classifier, (generated by all nodes), the `token` & `type` prediction loss (generated by only `token` or `type` nodes), and the edge prediction loss(generated by only UNK nodes). The model optimizes all four loss parts simultaneously with constant weight so that AST MGABs can benefit from the four subtasks.

4 EXPERIMENTAL SETUP

4.1 Datasets

We collect AST-level completion task-related datasets: **PY150K** and **JS150K**, which comprise parsed AST trees from sizable Python and JavaScript corpora, respectively. Each node in these trees is represented as a homogeneous graph with *type* and *value* attributes. To assess our approach's effectiveness against the Out-of-vocabulary (OOV) problem, we follow the conventions set by prior studies [18, 22, 24, 38, 40], restricting the word-table size to 1K, 10K, and 50K. Consequently, we generate six additional datasets: **PY1K**, **PY10K**, **PY50K**, **JS1K**, **JS10K**, and **JS50K**.

We employ different strategies to derive reference sets for the *JS* and *PY* datasets. We generate the retrieval model for JavaScript by

Table 1: Results of ReGCC and Baselines

	JS1K		JS10K		JS50K		PY1K		PY10K		PY50K	
	value	type	value	type	value	type	value	type	value	type	value	type
VanillaLSTM	53.2%	69.5%	58.0%	71.2%	59.7%	72.1%	50.0%	68.1%	52.7%	68.9%	53.7%	69.1%
ParentLSTM	56.5%	72.0%	61.5%	73.5%	63.4%	74.2%	52.6%	70.1%	55.9%	76.3%	56.9%	71.0%
PointerMixtureNet	56.5%	72.0%	62.3%	74.3%	64.1%	76.0%	53.0%	70.0%	56.9%	76.9%	57.2%	70.9%
Transformer	58.4%	73.3%	63.9%	74.8%	65.3%	75.9%	53.5%	70.6%	57.5%	71.5%	59.1%	71.9%
Transformer-XL	59.2%	72.1%	62.8%	74.1%	66.4%	76.2%	55.1%	72.5%	58.2%	73.2%	60.0%	72.4%
CCAG	62.8%	75.7%	66.7%	78.6%	68.2%	80.1%	61.9%	76.7%	63.2%	80.9%	64.2%	75.3%
CC-GGNN	64.1%	78.8%	67.9%	81.1%	70.5%	83.3%	63.3%	78.4%	65.7%	81.3%	66.9%	81.6%
OPT	67.8%	86.4%	75.6%	87.6%	79.0%	87.9%	62.1%	81.9%	67.4%	82.0%	68.9%	**82.3%**
ReGCC	**80.8%**	**87.4%**	**83.7%**	**87.8%**	**86.7%**	**90.0%**	**75.1%**	**82.0%**	**77.0%**	**82.1%**	**76.8%**	**82.3%**

using the *JS* code datasets from the research by Jain et al. [15], who have introduced code transformation techniques aimed at preserving various semantics, which are efficient for data augmentation. For the Python retrieval model, we adopt the method proposed by Lu et al. [26], which centers around generating positive examples while maintaining the semantics intact. We parse both corpora to their respective ASTs which are then utilized for training focused AST retrieval models measuring structural similarity (explained in Section 3.3). Following this, the retrieval model computes the AST representation for the entire datasets and identifies the most structurally similar examples[2] within the datasets with the same data field as the target AST. These identified structures then serve as a reference for the completion model.

4.2 Baselines

We select these baselines to compare with the ReGCC.

VanillaLSTM [18, 20, 33, 38] used LSTM to extract features from the spreading AST sequence. AST nodes and types are concatenated respectively as each temporal unit of LSTM, and the hidden state is passed through the prediction layer to derive the final prediction.

ParentLSTM [18] introduced the attention mechanism, which calculates the attention weight of the current hidden state and the previously hidden state within a context window as auxiliary information for the current state.

PointerMixtureNet [2, 18] proposed the first solution to the OOV problem. It can copy the context: choose to generate a token in the generative word list or copy a token from the context when generating a token.

Transformer [16, 30] is a powerful text generation model based entirely on the attention mechanism to extract valid information from other AST nodes.

Transformer-XL [23] proposed the first solution to the long-distance dependence problem in AST completion. A flat AST sequence is modeled, and the path from the predicted node to the root in the AST is fed to the BiLSTM to capture structural information.

CCAG [38] is the first AST completion scheme based entirely on GNN. It includes triple attention AST Gab, which obtains information from neighbor nodes, global and parent-child nodes, respectively.

[2]For a fair comparison with the baseline model, identical structures from the same dataset are retrieved instead of similar AST structures from a comprehensive corpus to avoid introducing additional information.

CC-GGNN [40] models AST graphs by separating the nodes to be predicted with different parent nodes, thereby narrowing down the range of candidate values. It mainly uses GGNN and self-attention to propagate information.

OPT [44] is the latest open-source large pre-trained language model, with its 175B version performance comparable to GPT-3 [5]. We take it as a representative of large language models to evaluate whether they can handle AST completion well.

4.3 Setup

We set that each model has an AST MGAB comprising 6 layers, with $n_{head} = 8$ employed during the multi-head attention calculation. The dimensions of all intermediate vectors and embedding layers are set to 512, with a drop-out rate of 0.1. In all experiments, we set the maximum subgraph length to 600. Any AST graph exceeding this length will be partitioned into subgraphs.

We bucket the dataset based on semantics to train the retrieval model, grouping instances with similar semantics into the same bucket. We set the number of iterations per epoch to equal the number of buckets, ensuring that every type of semantics is sampled. We sample two instances from the target bucket during each iteration and a negative example from a randomly chosen bucket. We do not design an additional validation stage for the retrieval model, as our objective is to retrieve similar code that complements the target code rather than clone detection. We save the retrieval model when the loss value decreases by less than 0.01 between two epochs. The learning rate for the retrieval model is set to 1e-4, and we employ the Adam optimization strategy.

For the completion model described in Section 3.4, we calculate all nodes during an iteration and sequentially predict the next node for each one. We set the maximum number of epochs to 10 and stop early if no improvement is observed. For all completion experiments, we set the initial learning rate to 6e-5 and decay it by a rate of 0.9 per epoch.

Our method models ASTs as heterogeneous graphs, but to maintain consistency with previous methods, we choose to convert the results back to homogeneous graphs during the validation. A unique mapping exists between heterogeneous and homogeneous code graphs (Section 3.1). In homogeneous graphs, we compute the accuracy for *type* and *value*.

Table 2: Results of Ablation Study

	JS1K		JS10K		JS50K		PY1K		PY10K		PY50K	
	value	type	value	type	value	type	value	type	value	type	value	type
ReGCC	**80.8%**	**87.4%**	**83.7%**	**87.8%**	**86.7%**	**90.0%**	**75.1%**	**82.0%**	**77.0%**	**82.1%**	**76.8%**	**82.3%**
-na	77.8%	84.2%	80.5%	84.6%	81.7%	85.2%	72.1%	79.0%	72.8%	79.3%	71.0%	79.2%
-gma	75.1%	81.1%	76.3%	80.7%	77.5%	82.2%	69.9%	76.2%	69.4%	76.7%	70.3%	77.9%
-ra	73.9%	80.2%	79.4%	83.4%	75.2%	80.6%	72.5%	79.1%	72.4%	79.2%	73.7%	80.3%
-ep	73.5%	**87.4%**	80.5%	**87.8%**	85.1%	**90.0%**	67.1%	**82.0%**	72.1%	**82.1%**	73.4%	**82.3%**

We copy the results of other baselines from their respective papers, except for OPT, for which we conducted additional experiments to evaluate its performance for AST completion. We fine-tuned OPT-125M on the completion dataset, added the *type* and *value* embeddings together as a single input, sequentially predicted the representation of the next node, and then passed them through linear layers to predict the values of *token* and *value* separately.

Our experiments were conducted on a machine with two GeForce RTX 3090 GPUs, an Intel(R) Xeon(R) Silver 4210R CPU @ 2.40GHz, and 125GB RAM, or on a machine with two NVIDIA Tesla V100 SXM2 32GB GPUs, six Intel(R) Xeon(R) Gold 6248 CPUs @ 2.50GHz, and 376GB RAM.

5 RESULTS AND ANALYSIS

5.1 Overall Performance

Firstly, to be consistent with the baselines, we converted the prediction results of ReGCC back to homogeneous graphs. We compared the prediction accuracy of ReGCC with all other baselines in terms of *value* and *type* on each dataset, and the results are shown in Table 1.

As demonstrated by the outcomes in Table 1, ReGCC emerges superior to current best-performing methods in both *value* prediction and *type* accuracy. Comparatively, when juxtaposed with graph-based CCAG and CC-GGNN, ReGCC exhibits a slight edge in *type* prediction whilst considerably surpassing them in *value* prediction. In particular, on **JS1K**, ReGCC's *value* accuracy shows a significant leap of 16.7% over the top-performing model, CC-GGNN — the largest such advancement noted. The vocabulary size in this dataset remains relatively minute, thereby increasing the prominence of the Out-of-vocabulary (OOV) problem. However, CCAG and CC-GGNN have not effectively addressed the OOV issue. This advancement implies that ReGCC can address the OOV problem, especially when operating under a limited vocabulary. Besides, ReGCC outshines the latest pre-trained language model, OPT, suggesting that extensive language models do not invariably hold an upper hand in recognizing and generating structural patterns. Overall, we note that our proposed ReGCC sets a new benchmark in AST completion tasks.

5.2 Ablation Study

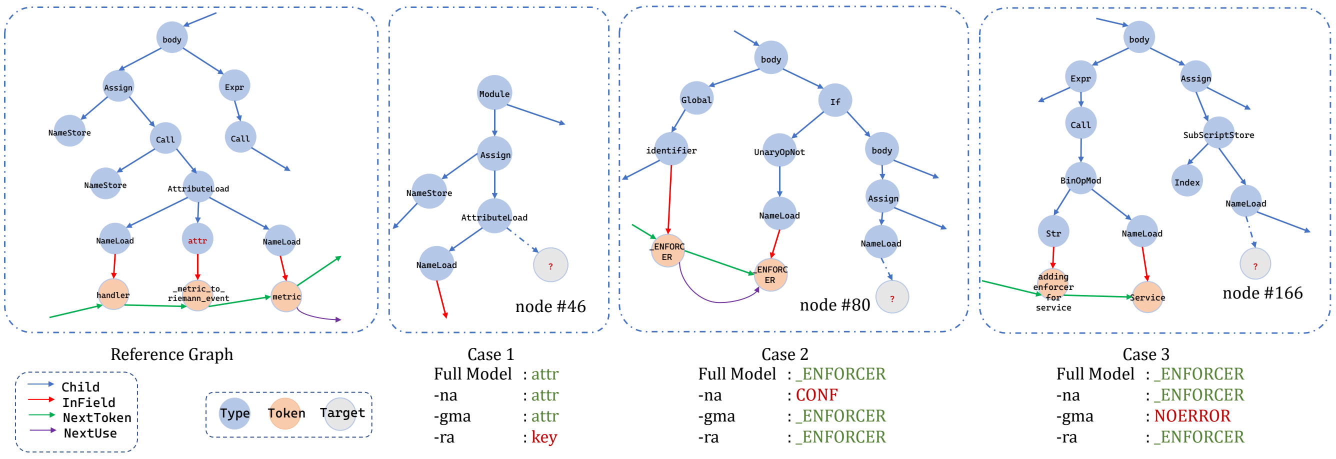
We performed a series of ablation studies on ReGCC, selectively disabling key components to assess their impact on the model's performance. We focused on four main components: three types of attention - neighborhood attention (NA), global and memory attention (GMA), and reference attention (RA) - incorporated in the Stacked Multi-Graph Attention Blocks (MGAB) module and edge prediction (EP), which addresses the Out-Of-Vocabulary (OOV) problem. To affirm the importance of these components, each one was systematically disabled, followed by retraining the model on all available datasets. Experimental outcomes, displayed in Table 2, confirm these components' paramount role in attaining ReGCC's high performance.

The results presented in Table 2 show a decline in both *value* and *type* accuracy across all datasets when any of the four components are deactivated, demonstrating their collective contribution to ReGCC's performance. For the three attention layers within the MGAB module, all three types of attention are instrumental in ensuring nodes procure adequate information. Among these, the global and memory attention layer is the most pivotal, as removing it leads to the most substantial accuracy decline. Neighborhood attention is important and integral to preserving graph structure specifics. Deactivating the reference attention also leads to performance degradation, highlighting the efficacy of reference information in predicting target graph nodes and endorsing the effectiveness of our retrieval model in providing appropriate references. Furthermore, while edge prediction does not influence *type* nodes, it significantly bolsters the accuracy of the *value*, particularly in datasets with smaller vocabularies. In the **JS1K** dataset, edge prediction boosted the *value* accuracy by 7.5%. This implies that edge prediction carries greater effectiveness in addressing the OOV issue within a small vocabulary.

5.3 Case Study

To delve deeper into ReGCC's effectiveness, we choose several illustrative examples and dissect the results predicted by ReGCC before and after ablation, as displayed in Fig. 7. We select a graph from the **PY10K** dataset, with the reference graph shown on the left representing a similar graph retrieved by the retrieval model based on the target. Case 1, Case 2, and Case 3 originate from different stages of the target graph generation, where the gray nodes (numbered #46, #80, and #166) are predicted based on preceding nodes. We then present the results from four scenarios: the original model, model ablated neighborhood attention (-na), model ablated global & memory attention (-gma), and model ablated reference attention (-ra). Correct predictions are indicated in green, while incorrect predictions are highlighted in red.

As per Figure 7, the intact ReGCC model accurately predicts in all cases, whereas the models lacking certain components fail to predict all cases. Further inspection reveals that the various attention layers function distinctly under different circumstances. The completion



Case 1	
Full Model	: attr
-na	: attr
-gma	: attr
-ra	: key

Case 2	
Full Model	: _ENFORCER
-na	: CONF
-gma	: _ENFORCER
-ra	: _ENFORCER

Case 3	
Full Model	: _ENFORCER
-na	: _ENFORCER
-gma	: NOERROR
-ra	: _ENFORCER

Figure 7: Results of some qualitative examples before and after ablation.

in Case 1 occurs at the inception of the graph when the available information is scarce, rendering the model incapable of predicting accurately without reference attention. However, the reference graph, mirroring a similar *AttributeLoad - attr* structure, can be exploited by the complete ReGCC to generate a correct prediction. In Case 2, the target token *_ENFORCER* is invoked continuously and repetitively. Here, the neighborhood attention layer assists the model in concentrating on the most recently appeared tokens, leading to an accurate prediction. The model's focus is dispersed throughout the entire graph on deactivating the neighborhood attention. Case 3, however, exhibits the converse effect where the target token *_ENFORCER* has a distant date of last usage (more than 80 nodes prior), creating a long-distance dependency. The absence of the global and memory attention layer makes it challenging for the model to detect the remotely-placed target. This substantiates the efficacy of the global and memory attention layer in resolving long-distance dependencies under such circumstances. Altogether, these case studies reinforce the value of every component of ReGCC, underlining their indispensable role in ably responding to varying situations, as anticipated.

6 DISSCUSION

Our research's effectiveness may encounter both internal and external threats.

Internally, the validity of our work is potentially threatened due to a lack of independent validation for the retrieval model. We posit that there is no need to evaluate the code clone detection performance of the ReGCC's retrieval model for two reasons. Firstly, our central research question does not pertain to code clone detection. The retrieval model is merely purposed to aid the completion model. As substantiated in section 5.2, Reference Attention materially enhances ReGCC's completion capacity, thereby testifying to the high utility of the ASTs fetched by our model. Secondly, our goal was to optimize the retrieval model's contribution to the completion model, compelling us to adopt a congruous structure, namely, the MGAB. Even though this structure may not yield the best results for clone retrieval, it effectively mirrors the information propagation pattern within the completion model, allowing for the transfer of parameters as initialization weights and thereby bolifying the stability of the completion model training.

Externally, the absence of testing on large language models poses a significant threat. We deployed a compact version of OPT, dubbed OPT-125M, to fine-tune the AST completion task, partly reflecting the efficacy of large language models. However, we refrained from evaluating larger language models like ChatGPT or GPT-3, which are accessible solely via paid APIs and entail considerable costs. Nonetheless, we substantiate our belief that large language models hold no distinctive edge over AST completion. This notion is based on the understanding that flattened sequences of values and types lack readability, thus restraining their ability to reap benefits from natural language corpus.

7 CONCLUSION

This study introduces ReGCC, a groundbreaking approach for AST-level code completion, seamlessly incorporating developers' cloning behavior, addressing out-of-vocabulary (OOV) snag, and considering node dependencies extending beyond context.

ReGCC has two models: a retrieval model and a completion model. For them, we propound the Multi-Graph Attention Block (MGAB), embracing three attention variants: neighborhood attention, global & memory attention, and reference attention. Neighborhood attention aims at preserving the structural heterogeneity of the graph, while global & memory attention deals with long-distance node dependencies. With its knack for aiding in the fabrication of target ASTs resonating with similar code structures, reference attention profoundly bolsters our model. Moreover, we have crafted an edge prediction strategy to grapple with the OOV predicament. Performances display our method's superiority over extant state-of-the-art AST completion models and the latest pre-trained language models in terms of both *value* and *type* accuracy.

Looking to future research, we anticipate enlarging in two principal realms. Primarily, we aspire to extract references not solely from the dataset itself but also from peripheral corpora. Secondarily, we intend to develop a cross-lingual universal retrieval and completion model. Acknowledging the potential importance of broad, universal, cross-lingual models, we plan to delve further into cross-lingual retrieval across diverse languages to nurture more efficient completion models.

ACKNOWLEDGMENTS

This work was sponsored by CCF-Tencent Rhino-Bird Open Research Fund (No. RAGR20230109). Besides, this work was carried out in part using computing resources at the High Performance Computing Center of Central South University.

REFERENCES

[1] Brenda S Baker. 2007. Finding clones with dup: Analysis of an experiment. *IEEE Transactions on Software Engineering* 33, 9 (2007), 608–621.

[2] Avishkar Bhoopchand, Tim Rocktäschel, Earl Barr, and Sebastian Riedel. 2016. Learning python code suggestion with a sparse pointer network. *arXiv preprint arXiv:1611.08307* (2016).

[3] Marc Brockschmidt, Miltiadis Allamanis, Alexander L. Gaunt, and Oleksandr Polozov. 2019. Generative Code Modeling with Graphs. In *International Conference on Learning Representations*. https://openreview.net/forum?id=Bke4KsA5FX

[4] Michael M. Bronstein, Joan Bruna, Yann LeCun, Arthur Szlam, and Pierre Vandergheynst. 2016. Geometric deep learning: going beyond Euclidean data. *CoRR* abs/1611.08097 (2016). arXiv:1611.08097 http://arxiv.org/abs/1611.08097

[5] Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. 2020. Language models are few-shot learners. *Advances in neural information processing systems* 33 (2020), 1877–1901.

[6] Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, et al. 2021. Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374* (2021).

[7] Matteo Ciniselli, Nathan Cooper, Luca Pascarella, Antonio Mastropaolo, Emad Aghajani, Denys Poshyvanyk, Massimiliano Di Penta, and Gabriele Bavota. 2021. An Empirical Study on the Usage of Transformer Models for Code Completion. *IEEE Transactions on Software Engineering* (2021).

[8] Matteo Ciniselli, Nathan Cooper, Luca Pascarella, Denys Poshyvanyk, Massimiliano Di Penta, and Gabriele Bavota. 2021. An empirical study on the usage of BERT models for code completion. In *2021 IEEE/ACM 18th International Conference on Mining Software Repositories (MSR)*. IEEE, 108–119.

[9] Zihang Dai, Zhilin Yang, Yiming Yang, Jaime Carbonell, Quoc V Le, and Ruslan Salakhutdinov. 2019. Transformer-xl: Attentive language models beyond a fixed-length context. *arXiv preprint arXiv:1901.02860* (2019).

[10] GitHub. 2021. Copilot. https://github.com/features/copilot

[11] Daya Guo, Alexey Svyatkovskiy, Jian Yin, Nan Duan, Marc Brockschmidt, and Miltiadis Allamanis. 2021. Learning to complete code with sketches. In *International Conference on Learning Representations*.

[12] Weihua Hu, Bowen Liu, Joseph Gomes, Marinka Zitnik, Percy Liang, Vijay Pande, and Jure Leskovec. 2019. Strategies for pre-training graph neural networks. *arXiv preprint arXiv:1905.12265* (2019).

[13] Ziniu Hu, Yuxiao Dong, Kuansan Wang, and Yizhou Sun. 2020. Heterogeneous graph transformer. In *Proceedings of the web conference 2020*. 2704–2710.

[14] Maliheh Izadi, Roberta Gismondi, and Georgios Gousios. 2022. CodeFill: Multi-Token Code Completion by Jointly Learning from Structure and Naming Sequences. In *Proceedings of the 44th International Conference on Software Engineering* (Pittsburgh, Pennsylvania) *(ICSE '22)*. Association for Computing Machinery, New York, NY, USA, 401–412. https://doi.org/10.1145/3510003.3510172

[15] Paras Jain, Ajay Jain, Tianjun Zhang, Pieter Abbeel, Joseph E. Gonzalez, and Ion Stoica. 2020. Contrastive Code Representation Learning. *arXiv preprint* (2020).

[16] Seohyun Kim, Jinman Zhao, Yuchi Tian, and Satish Chandra. 2021. Code Prediction by Feeding Trees to Transformers. In *Proceedings of the 43rd International Conference on Software Engineering* (Madrid, Spain) *(ICSE '21)*. IEEE Press, 150–162. https://doi.org/10.1109/ICSE43902.2021.00026

[17] Jian Li, Yue Wang, Michael R Lyu, and Irwin King. 2017. Code completion with neural attention and pointer networks. *arXiv preprint arXiv:1711.09573* (2017).

[18] Jian Li, Yue Wang, Michael R Lyu, and Irwin King. 2017. Code completion with neural attention and pointer networks. *arXiv preprint arXiv:1711.09573* (2017).

[19] Xiang Ling, Lingfei Wu, Saizhuo Wang, Gaoning Pan, Tengfei Ma, Fangli Xu, Alex X Liu, Chunming Wu, and Shouling Ji. 2021. Deep graph matching and searching for semantic code retrieval. *ACM Transactions on Knowledge Discovery from Data (TKDD)* 15, 5 (2021), 1–21.

[20] Chang Liu, Xin Wang, Richard Shin, Joseph E Gonzalez, and Dawn Song. 2016. Neural code completion. (2016).

[21] Fang Liu, Zhiyi Fu, Ge Li, Zhi Jin, Hui Liu, and Yiyang Hao. 2022. Non-autoregressive Model for Full-line Code Completion. *arXiv preprint arXiv:2204.09877* (2022).

[22] Fang Liu, Ge Li, Bolin Wei, Xin Xia, Zhiyi Fu, and Zhi Jin. 2020. A Self-Attentional Neural Architecture for Code Completion with Multi-Task Learning. In *Proceedings of the 28th International Conference on Program Comprehension* (Seoul, Republic of Korea) *(ICPC '20)*. Association for Computing Machinery, New York, NY, USA, 37–47. https://doi.org/10.1145/3387904.3389261

[23] Fang Liu, Ge Li, Bolin Wei, Xin Xia, Zhiyi Fu, and Zhi Jin. 2020. A self-attentional neural architecture for code completion with multi-task learning. In *Proceedings of the 28th International Conference on Program Comprehension*. 37–47.

[24] Fang Liu, Ge Li, Bolin Wei, Xin Xia, Zhiyi Fu, and Zhi Jin. 2022. A unified multi-task learning model for AST-level and token-level code completion. *Empirical Software Engineering* 27, 4 (2022), 1–38.

[25] Fang Liu, Ge Li, Yunfei Zhao, and Zhi Jin. 2021. Multi-Task Learning Based Pre-Trained Language Model for Code Completion. In *Proceedings of the 35th IEEE/ACM International Conference on Automated Software Engineering* (Virtual Event, Australia) *(ASE '20)*. Association for Computing Machinery, New York, NY, USA, 473–485. https://doi.org/10.1145/3324884.3416591

[26] Shuai Lu, Nan Duan, Hojae Han, Daya Guo, Seung-won Hwang, and Alexey Svyatkovskiy. 2022. ReACC: A Retrieval-Augmented Code Completion Framework. *arXiv preprint arXiv:2203.07722* (2022).

[27] Chanchal K Roy and James R Cordy. 2008. An empirical study of function clones in open source software. In *2008 15th Working Conference on Reverse Engineering*. IEEE, 81–90.

[28] Yixin Su, Rui Zhang, Sarah Erfani, and Zhenghua Xu. 2021. Detecting beneficial feature interactions for recommender systems. In *Proceedings of the AAAI conference on artificial intelligence*, Vol. 35. 4357–4365.

[29] Alexey Svyatkovskiy, Shao Kun Deng, Shengyu Fu, and Neel Sundaresan. 2020. IntelliCode Compose: Code Generation Using Transformer. In *Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering* (Virtual Event, USA) *(ESEC/FSE 2020)*. Association for Computing Machinery, New York, NY, USA, 1433–1443. https://doi.org/10.1145/3368089.3417058

[30] Alexey Svyatkovskiy, Shao Kun Deng, Shengyu Fu, and Neel Sundaresan. 2020. Intellicode compose: Code generation using transformer. In *Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. 1433–1443.

[31] Alexey Svyatkovskiy, Sebastian Lee, Anna Hadjitofi, Maik Riechert, Juliana Vicente Franco, and Miltiadis Allamanis. 2021. Fast and memory-efficient neural code completion. In *2021 IEEE/ACM 18th International Conference on Mining Software Repositories (MSR)*. IEEE, 329–340.

[32] Alexey Svyatkovskiy, Sebastian Lee, Anna Hadjitofi, Maik Riechert, Juliana Vicente Franco, and Miltiadis Allamanis. 2021. Fast and Memory-Efficient Neural Code Completion. In *2021 IEEE/ACM 18th International Conference on Mining Software Repositories (MSR)*. 329–340. https://doi.org/10.1109/MSR52588.2021.00045

[33] Alexey Svyatkovskiy, Ying Zhao, Shengyu Fu, and Neel Sundaresan. 2019. Pythia: Ai-assisted code completion system. In *Proceedings of the 25th ACM SIGKDD international conference on knowledge discovery & data mining*. 2727–2735.

[34] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. *Advances in neural information processing systems* 30 (2017).

[35] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Lio, and Yoshua Bengio. 2017. Graph attention networks. *arXiv preprint arXiv:1710.10903* (2017).

[36] Wenhan Wang, Sijie Shen, Ge Li, and Zhi Jin. 2020. Towards Full-line Code Completion with Neural Language Models. https://doi.org/10.48550/ARXIV.2009.08603

[37] Wenhan Wang, Sijie Shen, Ge Li, and Zhi Jin. 2020. Towards Full-line Code Completion with Neural Language Models. *arXiv preprint arXiv:2009.08603* (2020).

[38] Yanlin Wang and Hui Li. 2021. Code completion by modeling flattened abstract syntax trees as graphs. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 35. 14015–14023.

[39] Zhirong Wu, Yuanjun Xiong, Stella X Yu, and Dahua Lin. 2018. Unsupervised feature learning via non-parametric instance discrimination. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 3733–3742.

[40] Kang Yang, Huiqun Yu, Guisheng Fan, Xingguang Yang, and Zijie Huang. 2022. A graph sequence neural architecture for code completion with semantic structure features. *Journal of Software: Evolution and Process* 34, 1 (2022), e2414.

[41] Dongjin Yu, Quanxin Yang, Xin Chen, Jie Chen, and Yihang Xu. 2023. Graph-based code semantics learning for efficient semantic code clone detection. *Information and Software Technology* 156 (2023), 107130.

[42] Fengji Zhang, Bei Chen, Yue Zhang, Jin Liu, Daoguang Zan, Yi Mao, Jian-Guang Lou, and Weizhu Chen. 2023. RepoCoder: Repository-Level Code Completion Through Iterative Retrieval and Generation. *arXiv preprint arXiv:2303.12570* (2023).

[43] Jiawei Zhang, Haopeng Zhang, Congying Xia, and Li Sun. 2020. Graph-bert: Only attention is needed for learning graph representations. *arXiv preprint arXiv:2001.05140* (2020).
[44] Susan Zhang, Stephen Roller, Naman Goyal, Mikel Artetxe, Moya Chen, Shuohui Chen, Christopher Dewan, Mona Diab, Xian Li, Xi Victoria Lin, et al. 2022. Opt: Open pre-trained transformer language models. *arXiv preprint arXiv:2205.01068* (2022).
[45] Yue Zou, Bihuan Ban, Yinxing Xue, and Yun Xu. 2020. CCGraph: a PDG-based code clone detector with approximate graph matching. In *Proceedings of the 35th IEEE/ACM International Conference on Automated Software Engineering*. 931–942.