Appendix

1 Code Summarization

1.1 Architecture

The architecture of CoCoAST (Fig. 1) on code summarization follows the general Seq2Seq framework and includes three major components: an AST encoder, a code token encoder, and a summary decoder. Given an input method, the AST encoder captures the semantic and structural information of its AST. The code token encoder encodes the lexical information of the method. The decoder integrates the multi-channel representations from the two encoders and incorporates a copy mechanism [1] to generate the code summary The AST and code encoder have been introduced in Section 3. Similar to the code token encoder, we adopt Transformer as the backbone of the decoder. Unlike the original decoder module in [2], we need to integrate two encoding sources from code and AST encoders. The serial strategy [3] is adopted, which is to compute the encoder-decoder attention one by one for each input encoder (Fig. 2). In each cross-attention layer, the encoding of ASTs ($\mathbf{h}^{(a)} = (\mathbf{h}_\mathbf{1}^{(\mathbf{a})}, ..., \mathbf{h}_\mathbf{l}^{(\mathbf{a})})$ flatted by preorder traversal) or codes ($\mathbf{o} = (\mathbf{o_1}, ..., \mathbf{o_n})$) is queried by the output of the preceding summary self-attention $\mathbf{s} = (\mathbf{s_1}, ..., \mathbf{s_m})$.

$$
\begin{aligned}
\mathbf{z}_i &= \sum_{j=1}^{n} \alpha_{ij} \left(\mathbf{W^V} \mathbf{h}_j^{(a)}\right), \alpha_{ij} = \frac{\exp e_{ij}^{ast}}{\sum_{k=1}^{n} \exp e_{ik}^{ast}} \\
\mathbf{y}_i &= \sum_{j=1}^{n} \alpha_{ij} \left(\mathbf{W^V} \mathbf{o}_j\right), \alpha_{ij} = \frac{\exp e_{ij}^{code}}{\sum_{k=1}^{n} \exp e_{ik}^{code}} \\
e_{ij}^{ast} &= \frac{(\mathbf{W_d^Q} \mathbf{s}_i)^T \left(\mathbf{W_d^K} \mathbf{h}_j^{(a)}\right)}{\sqrt{d_k}}, e_{ij}^{code} = \frac{(\mathbf{W_d^Q} \mathbf{z}_i)^T \left(\mathbf{W_d^K} \mathbf{o}_j\right)}{\sqrt{d_k}}
\end{aligned}
\tag{1}
$$

where $\mathbf{W_d^Q}$, $\mathbf{W_d^K}$ and $\mathbf{W_d^V}$ are trainable projection matrices for queries, keys and values. l is the number of subtrees. m and n are the length of code and summary tokens, respectively. Following [2], we adopt a multi-head attention mechanism in the self-attention and cross-attention layers of the decoder. After stacking several decoder layers, we add a softmax operator to obtain the generation probability $P_t^{(g)}$ of each summary token.

We further incorporate the copy mechanism [1] to enable the decoder to copy rare tokens directly from the input codes. This is motivated by the fact that many tokens (about 28% in the Funcom dataset) are directly copied from the source code (e.g., function names and variable names) in the summary. Specifically, we learn a copy probability through an attention layer:

$$
P_t^{(c)}(i) = \frac{\exp(< \mathbf{W^{cp}} \mathbf{h}_i^{(c)}, \mathbf{h}_t^{(s)} >)}{\sum_{k=1}^{T_c} \exp(< \mathbf{W^{cp}} \mathbf{h}_k^{(c)}, \mathbf{h}_t^{(s)} >)}, \tag{2}
$$

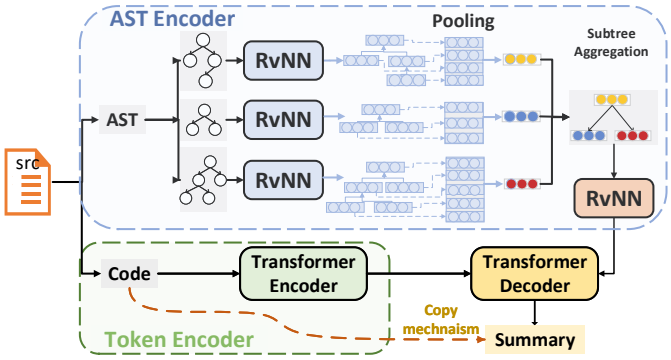

Fig. 1. The framework of CoCoAST on code summarization.

where $P_t^{(c)}(i)$ is the probability for choosing the i-th token from source code in the summary position t, $\mathbf{h}_i^{(c)}$ is the encoding vector of the i-th code token, $\mathbf{h}_t^{(s)}$ is the decoding vector of the t-th summary token, $\mathbf{W^{cp}}$ is a learnable projection matrix to map $h_i^{(c)}$ to the space of $h_i^{(s)}$, and T_c is the code length. The final probability for selecting the token w as t-th summary token is defined as:

$$
P_t(w) = \gamma_t P_t^{(g)}(w) + (1 - \gamma_t) \sum\nolimits_{i:w_i^{(c)}=w} P_t^{(c)}(i), \tag{3}
$$

where $w_i^{(c)}$ is the i-th code token and γ_t is a learned combination probability defined as $\gamma_t = \text{sigmoid}(\Gamma(\mathbf{h}_t^{(s)}))$, where Γ is a feed forward neural network. Finally, we use Maximum Likelihood Estimation as the objective function and apply AdamW for optimization.

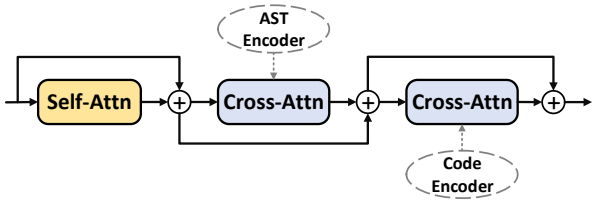

Fig. 2. The serial strategy for integrating two encoding sources in the decoder.

1.2 Hyperparameters

Table 1 summarizes the hyperparameters used in our experiments. len_{code} and len_{sum} are the sequence length of code and summary, respectively. Each covers at least 90% of the training set. $vocab_{code}$, $vocab_{sum}$, and $vocab_{ast}$ are the vocabulary size of code, summary, and AST. len_{pos} refers

to the clipping distance in relative position. d_{Emb} is the dimension of the embedding layer. $heads$ and $layers$ indicate the number of layers and heads in Transformer, respectively. d_{ff} is the hidden layer dimension of feed-forward in Transformer. d_{RvNN} and $activate_f$ are dimension of hidden state and activate function in RvNN, respectively. $layer_{ff}$ is the number of the feed-forward layer in the copy component.

The values of these hyperparameters are set according to the related work [4], [5]. We adjust the sizes of d_{Emb}, d_{ff}, and d_{RvNN} empirically. The batch size is set according to the computing memory.

TABLE 1
Hyperparameters in our experiments on code summarization.

	Parameter	TL-CodeSum	Funcom
Data	len_{code}	150	110
	len_{sum}	50	13
	$vocab_{code}$	50000	50000
	$vocab_{sum}$	30000	30000
	$vocab_{ast}$	10000	10000
Embedding	d_{Emb}	640	512
	len_{pos}	32	32
Transformer	d_{ff}	4096	2048
	$heads$	4	8
	$layers$	6	6
RvNN	d_{RvNN}	640	512
	$activate_f$	relu	relu
Copy	$layer_{ff}$	1	1
Training	$drop\ out$	0.2	0.2
	$optimizer$	Adam	Adam
	$batch\ size$	128	128
	$learning\ rate$	0.0001	0.0001

1.3 Runtime of Different AST Representation Models

Tab2 and Table 3 show the time cost for differenct approaches on the two datasets. The 2nd column is the training time of one epoch. The 3rd column is the total training time. And the last column is the inference time per function. On the Funcom dataset, the baselines have different training time. Most of them range from 7 to 69 hours (except that Hybrid-DRL takes 633 hours). The inference time per function is 0.0073s for CASTS and 0.001 to 0.0139s for other baselines. Our approach has comparable time cost as the baselines. Similar performance can also be observed on TL-CodeSum. During training, HybridDrl firstly trains hybrid code representation by applying LSTM and Tree-LSTM to code token and AST. Then it generates summary based on actor-critic reinforcement learning. Therefore, it takes long time for HybridDrl to train the model.

1.4 Experimental Results on Deduplicated Dataset

In our experiment, we find the existence of code duplication in TL-CodeSum: around 20% code snippets in the testing set can be found in the training set. Thus, we remove the duplicated samples from the testing set and re-evaluate all approaches. Table 4 shows the result of different models in the rest testings set without duplicated samples. Our model still outperforms all baselines.

TABLE 2
Time cost of different AST representation models in Funcom on code summariztion.

Model	train/epoch	train/total	infer
Code2seq	41m48s	69h40m00s	0.0010s
HybridDrl	21h06m43s	633h21m30s	0.0139s
Astattgru	10m46s	07h10m40s	0.0037s
CodeAstnn	53m58s	44h04m22s	0.0090s
CoCoAST	43m24s	26h02m24s	0.0073s

TABLE 3
Time cost of different AST representation models in TL-CodeSum on code summarization.

Model	train/epoch	train/total	infer
Code2seq	01m51s	06h10m00s	0.0021s
HybridDrl	1h31m41s	45h50m30s	0.0514s
Astattgru	01m40s	06h36m40s	0.0073s
CodeAstnn	07m58s	26h33m20s	0.0269s
CoCoAST	07m45s	25h50m00s	0.0358s

1.5 Evaluation Metrics

We provide the details of the evaluation metrics we used in the experiments.

1.5.1 BLEU

BLEU measures the average n-gram precision between the reference sentences and generated sentences, with brevity penalty for short sentences. The formula to compute BLEU-1/2/3/4 is:

$$\text{BLEU-N} = BP \cdot \exp \sum_{n=1}^{N} \omega_n \log p_n, \tag{4}$$

where p_n (n-gram precision) is the fraction of n-grams in the generated sentences which are present in the reference sentences, and ω_n is the uniform weight 1/N. Since the generated summary is very short, high-order n-grams may not overlap. We use the +1 smoothing function [6]. BP is brevity penalty given as:

$$BP = \begin{cases} 1 & \text{if } c > r \\ e^{(1-r/c)} & \text{if } c \leq r \end{cases} \tag{5}$$

Here, c is the length of the generated summary, and r is the length of the reference sentence.

1.5.2 ROUGE-L

Based on longest common subsequence (LCS), ROUGE-L is widely used in text summarization. Instead of using only recall, it uses F-score which is the harmonic mean of precision and recall values. Suppose A and B are generated and reference summaries of lengths c and r respectively, we have:

$$\begin{cases} P_{ROUGE-L} = \frac{LCS(A,B)}{c} \\ R_{ROUGE-L} = \frac{LCS(A,B)}{r} \end{cases} \tag{6}$$

$F_{ROUGE-L}$, which indicates the value of ROUGE-L, is calculated as the weighted harmonic mean of $P_{ROUGE-L}$ and $R_{ROUGE-L}$:

$$F_{ROUGE-L} = \frac{(1+\beta^2)\, P_{ROUGE-L} \cdot R_{ROUGE-L}}{R_{ROUGE-L} + \beta^2 P_{ROUGE-L}} \tag{7}$$

β is set to 1.2 as in [7], [8].

TABLE 4
Performance of different models on deduplicated TL-CodeSum dataset.

Model	BLEU	Meteor	Rouge-L	Cider
CodeNN	11.04	7.57	20.61	0.51
HDeepcom	10.58	7.18	20.23	0.45
Attgru	12.03	7.94	22.25	0.55
Astattgru	14.37	9.47	26.01	0.78
HybridDrl	13.27	7.26	23.47	0.61
Code2seq	13.91	7.62	21.44	0.42
CodeAstnn	23.94	14.91	36.92	1.68
NCS	23.46	15.12	37.42	1.66
CoCoAST	**27.13**	**17.14**	**39.97**	**2.03**

1.5.3 METEOR

METEOR is a recall-oriented metric that measures how well the model captures the content from the references in the generated sentences and has a better correlation with human judgment. Suppose m is the number of mapped unigrams between the reference and generated sentence with lengths c and r respectively. Then, precision, recall and F are given as:

$$P = \frac{m}{c},\ R = \frac{m}{r},\ F = \frac{PR}{\alpha P + (1-\alpha)R} \tag{8}$$

The sequence of mapping unigrams between the two sentences is divided into the fewest possible number of "chunks". This way, the matching unigrams in each "chunk" are adjacent (in two sentences) and the word order is the same. The penalty is then computed as:

$$\text{Pen} = \gamma \cdot \text{ frag }^{\beta} \tag{9}$$

where frag is a fragmentation fraction: frag $= ch/m$, where ch is the number of matching chunks and m is the total number of matches. The default values of α, β, γ are 0.9, 3.0 and 0.5 respectively.

1.5.4 CIDER

CIDER is a consensus-based evaluation metric used in image captioning tasks. The notions of importance and accuracy are inherently captured by computing the TF-IDF weight for each n-gram and using cosine similarity for sentence similarity. To compute CIDER, we first calculate the TF-IDF weighting $g_k(s_i)$ for each n-gram ω_k in reference sentence s_i. Here ω is the vocabulary of all n-grams. Then we use the cosine similarity between the generated sentence and the reference sentences to compute CIDER_n score for n-grams of length n. The formula is given as:

$$\text{CIDER}_n\left(c_i, s_i\right) == \frac{< \boldsymbol{g^n}\left(c_i\right), \boldsymbol{g^n}\left(s_i\right) >}{\left\|\boldsymbol{g}^n\left(c_i\right)\right\| \left\|\boldsymbol{g}^n\left(s_i\right)\right\|} \tag{10}$$

where $\boldsymbol{g^n}(s_i)$ is a vector formed by $g_k(s_i)$ corresponding to all the n-grams (n varying from 1 to 4). c_i is the i^{th} generated sentence. Finally, the scores of various n-grams can be combined to calculate CIDER as follows:

$$\text{CIDER}\left(c_i, s_i\right) = \sum_{n=1}^{N} w_n \text{CIDER}_n\left(c_i, s_i\right) \tag{11}$$

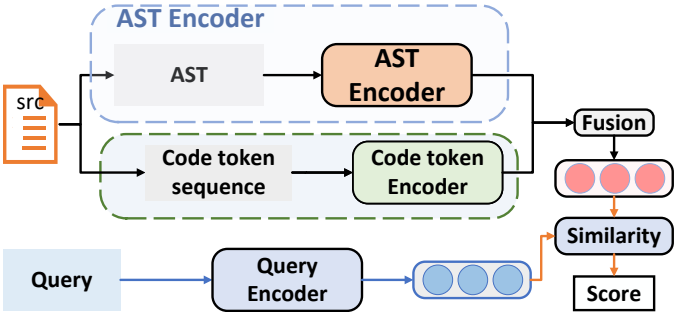

Fig. 3. The framework of CoCoAST on code search.

1.6 Human evaluation

We conduct a human evaluation to evaluate the effectiveness of the summaries generated by our approach CoCoAST and the other three approaches. The shows that CoCoAST outperforms the others in all three aspects: similarity, naturalness, and informativeness. we confirmed the dominance of our approach using Wilcoxon signed-rank tests for human evaluation. The result shown in Table 5 reflects that the improvement of CoCoAST over other approaches is statistically significant with all p-values smaller than 0.05 at 95% confidence level (except CodeAstnn in the naturalness).

2 Code Search

For code search task, the overall framework is shown in Fig. 3. We use fused code and AST representation as the final code representation. Similar to the code token encoder (introduce in Section 3.2), we use a transformer encoder as the query encoder to obtain the query embedding. We measure the semantic similarity of the code and query by the vector distance. Specifically, let $\mathbf{q}_{[\mathbf{cls}]}$ denote the query representation and $\mathbf{v}_{\mathbf{code}}$ denote the code representation. The semantic similarity between the query and code snippet is calculated as :

$$Similarity(code, query) =< \mathbf{v}_{\mathbf{code}} \cdot \mathbf{q}_{[\mathbf{cls}]} > \tag{12}$$

where$< \cdot >$ indicates dot product.

We use contrastive loss function named InfoNCE [9] to optimize the parameters of the model. Specifically, given a query $\mathbf{q_i}$, we denote the paired $\mathbf{code_i}$ as $\mathbf{code_i}^+$ and unpaired $\mathbf{code_k}$ as $\mathbf{code_k}^-$ ($i = 1, ..., bs$ and $k = 1, ..., bs$, bs is batch size.). For a query $\mathbf{q_i}$, the loss is calculated by:

$$L_{q_i} = -log\frac{exp < \mathbf{q}_{[\mathbf{cls}]_\mathbf{i}} \cdot \mathbf{v}^+_{\mathbf{code_i}}/\tau >}{exp < \mathbf{q}_{[\mathbf{cls}]_\mathbf{i}} \cdot \mathbf{v}^+_{\mathbf{code_i}})/\tau > + \sum_{k=1}^{bs} exp < \mathbf{q}_{[\mathbf{cls}]_\mathbf{i}} \cdot \mathbf{v}^-_{\mathbf{code_k}}/\tau >} \tag{13}$$

where τ is the temperature hyperparameter [10], [11] and is set to 0.07 following previous works [11], [12]. Intuitively, this optimization objective is to maximize the semantic similarity of the query and its paired code snippet and minimize the semantic similarity of the query and its unpaired code snippets. In the same way, for a code snippet $code_i$, the loss is calculated by:

$$L_{code_i} = -log\frac{exp < \mathbf{v}_{\mathbf{code_i}} \cdot \mathbf{q}^+_{[\mathbf{cls}]_\mathbf{i}}/\tau >}{exp < \mathbf{v}_{\mathbf{code_i}} \cdot \mathbf{q}^+_{[\mathbf{cls}]_\mathbf{i}})/\tau > + \sum_{k=1}^{bs} exp < \mathbf{v}_{\mathbf{code_i}} \cdot \mathbf{q}^-_{[\mathbf{cls}]_\mathbf{k}}/\tau >} \tag{14}$$

TABLE 5
Statistics significance p-value of CoCoAST over other methods in human evaluation.

Model	Informativeness	Naturalness	Similarity
CoCoAST	$1.83e^{-6}$	$5.15e^{-7}$	$8.71e^{-09}$
Astattgru	$5.23e^{-4}$	$7.49e^{-3}$	$3.53e^{-6}$
NCS	$1.34e^{-2}$	$4.66e^{-1}$	$2.86e^{-5}$

TABLE 6
Hyperparameters in our experiments on code search.

Data	len_{code}	256
	len_{sum}	128
	$vocab_{code}$	50,26
	$vocab_{sum}$	50,26
	$vocab_{ast}$	10000
Embedding	d_{Emb}	512
	len_{pos}	32
Transformer	d_{ff}	2048
	$heads$	8
	$layers$	6
RvNN	d_{RvNN}	512
	$activate_f$	relu
Training	$drop\ out$	0.2
	$optimizer$	Adam
	$batch\ size$	64
	$learning\ rate$	0.0002

where ${q_i}^+$ is the paired query of input code snippet $code_i$, and ${q_k}^-$ denotes the unpaired query. To this end, the overall contrastive learning loss function for a mini-batch is:

$$L = \sum_{i=1}^{bs}(L_{q_i} + L_{code_i}) \tag{15}$$

We apply AdamW [13] algorithm to optimize the loss functions.

2.1 Hyperparameters

Table 6 summarizes the hyperparameters used in our experiments on code search. The values of these hyperparameters are set according to the related work [5], [14]. We adjust the sizes of d_{Emb}, d_{ff}, and d_{RvNN} empirically. The batch size is set according to the computing memory.

REFERENCES

[1] A. See, P. J. Liu, and C. D. Manning, "Get to the point: Summarization with pointer-generator networks," in *ACL*, 2017.

[2] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, "Attention is all you need," in *NIPS*, 2017, pp. 5998–6008.

[3] J. Libovický, J. Helcl, and D. Mareček, "Input combination strategies for multi-source transformer decoder," in *WMT*, 2018.

[4] W. U. Ahmad, S. Chakraborty, B. Ray, and K. Chang, "A transformer-based approach for source code summarization," in *ACL*, 2020.

[5] 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 *ICSE*, 2019.

[6] C. Lin and F. J. Och, "Automatic evaluation of machine translation quality using longest common subsequence and skip-bigram statistics," in *ACL*. ACL, 2004, pp. 605–612.

[7] J. Zhang, X. Wang, H. Zhang, H. Sun, and X. Liu, "Retrieval-based neural source code summarization," in *ICSE*, 2020.

[8] Y. Wan, Z. Zhao, M. Yang, G. Xu, H. Ying, J. Wu, and P. S. Yu, "Improving automatic source code summarization via deep reinforcement learning," in *ASE*, 2018.

[9] A. v. d. Oord, Y. Li, and O. Vinyals, "Representation learning with contrastive predictive coding," 2018.

[10] Z. Wu, Y. Xiong, S. X. Yu, and D. Lin, "Unsupervised feature learning via non-parametric instance discrimination," in *CVPR*. Computer Vision Foundation / IEEE Computer Society, 2018, pp. 3733–3742.

[11] K. He, H. Fan, Y. Wu, S. Xie, and R. B. Girshick, "Momentum contrast for unsupervised visual representation learning," in *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition, CVPR 2020, Seattle, WA, USA, June 13-19, 2020*. Computer Vision Foundation / IEEE, 2020, pp. 9726–9735. [Online]. Available: https://doi.org/10.1109/CVPR42600.2020.00975

[12] P. Jain, A. Jain, T. Zhang, P. Abbeel, J. Gonzalez, and I. Stoica, "Contrastive code representation learning," in *EMNLP (1)*. Association for Computational Linguistics, 2021, pp. 5954–5971.

[13] I. Loshchilov and F. Hutter, "Decoupled weight decay regularization," in *ICLR*, 2019.

[14] H. Husain, H. Wu, T. Gazit, M. Allamanis, and M. Brockschmidt, "Codesearchnet challenge: Evaluating the state of semantic code search," *arXiv Preprint*, 2019. [Online]. Available: https://arxiv.org/abs/1909.09436