APPENDIX

A. Experiments Setup

Due to differences in the scale of parameters, training datasets, and other conditions, we will conduct a more detailed assessment between our method and Large Language Models (LLMs) that have 1 billion parameters in this appendix. Within the appendix, we have selected two representative large models, GPT-3.5-turbo-instruct and Code Llama 7B, which respectively symbolize general-purpose LLMs and code-oriented LLMs. Similar to the main text, we have included the complete experimental code and their results on the test set in our open-source link for the reference of other researchers.

1) The Prompt Template: The prompt template of the LLMs is as follows:

```
# You are an expert Java programmer, please describe the functionality of the
method in a short word(less than 40):

# Example code:
public synchronized boolean start() {
    if (!isStarted()) {
    final ExecutorService tempExec;
    executor = getExternalExecutor();
    if (executor == null) {
        executor = tempExec = createExecutor();
    }
    else {
        tempExec = null;}
        future = executor.submit(createTask(tempExec));
        return true;
    }
    return false;
}
# The comment is:
Starts the background initialization

# Test code:
public String function() {
    ...;
}
# Summary:
```

Fig. A1. Examples from the Case Study

2) Evaluation Metrics: The evaluation metrics of our experiments are as follows:

BLEU. The BLEU metric evaluates the quality of text by calculating the degree of overlap in n-grams between the machine-generated text and a set of reference texts. The calculation method is as follows:

$$\text{BLEU} - \text{N} = BP \cdot \exp \sum_{n=1}^{N} \omega_n \log p_n \tag{A1}$$

where $\log p_n$ refers to the precision of n-grams, n typically ranges from 1 to 4. ω_n represents the weight, which, in our experiments, is distributed equally among the different n-gram sizes. BP stands for the brevity penalty, which is a factor used to penalize short sentences. The calculation of this factor is as follows:

$$BP = \begin{cases} 1 \ , \ \text{if} \ \ c > r, \\ e^{1-r/c} \ , \ \text{if} \ \ c \leq r, \end{cases} \tag{A2}$$

where c represents the length of the generated text, and r represents the closest length of the reference text.

METEOR. The METEOR evaluation metric comprehensively considers precision, recall, and sequence similarity. The calculation method is as follows:

$$P = \frac{m}{c}, R = \frac{m}{r}, F = \frac{PR}{\alpha P + (1-\alpha)R} \tag{A3}$$

Considering word order information, the final METEOR score also incorporates a penalty. The purpose of the penalty is to account for the difference in word order between the generated text and the reference text. That is,

$$F_{\text{METEOR}} = (1 - \text{penalty}) \cdot F \tag{A4}$$

The penalty is determined based on the order of aligned words and the degree to which their order in the text mismatches. Specifically, the penalty can be computed as:

$$\text{penalty} = \gamma \cdot \text{frag}^{\beta} \tag{A5}$$

where frag is a fragmentation fraction, which can be defined as $\text{frag} = ch/m$, where ch is the number of matching chunks and m is the total number of matches.

ROUGE. ROUGE-L uses the Longest Common Subsequence (LCS) to evaluate the similarity between two sequences. This variant is particularly suitable for assessing sentence-level similarity between generated text and reference text. The score of ROUGE-L is based on recall, precision, and F-score, with the calculation formulas as follows:

$$\begin{aligned} P_{ROUGE-L} &= \frac{LCS(X,Y)}{c} \\ R_{ROUGE-L} &= \frac{LCS(X,Y)}{r} \\ F_{ROUGE-L} &= \frac{(1+\beta^2) P_{ROUGE-L} \cdot R_{ROUGE-L}}{R_{ROUGE-L} + \beta^2 P_{ROUGE-L}} \end{aligned} \tag{A6}$$

where $LCS(X,Y)$ is the length of the longest common subsequence between sequence X and sequence Y. $\text{len}(X)$ and $\text{len}(Y)$ are the lengths of sequences X and Y, respectively. $F_{ROUGE-L}$ is the F-score calculated based on LCS, and β is used to adjust the importance between recall and precision.

B. Model Details Comparison

Given the closed-source of ChatGPT, here we primarily compare or model with the open-source LLM Code Llama 7B.

TABLE A.I
MODEL DETAILS COMPARISION

Model	Dataset	Parameters	Inference Time	Infer
APIMsGSum	44 MB	120 M	37 m	0.25s
Code Llama	2T + 500 B	7 B	5 h 32 m	2.29s

Table A.I showcases the size of the training dataset, the scale of model parameters, and the inference time on the test set. It is evident that the parameter scale of Code Llama is approximately 60 times larger than our method, which consequently

TABLE A.II
OBJECTIVE METRICS EVALUATION

Model	BLEU	METEOR	ROUGE
APIMsGSum	50.37	34.38	62.82
Code Llama	25.08	28.16	24.82
GPT-3.5-turbo	33.61	26.49	33.39

makes its inference time nearly 10 times longer than ours. On the other hand, the smallest full-precision Code Llama 7B model occupies about 26GB of VRAM, which is challenging to support for the majority of developers using consumer-grade graphics cards due to the substantial computational resources required. The differences in deployment costs and inference time allow researchers with varying computational resources to select the appropriate model to assist in code annotation generation tasks based on their needs.

C. Objective Metrics Evaluation

Table A.II displays the results of APIMsGSum and two types of LLMs on the code summarization task, using several syntactic overlap metrics. Based on this, we can obtain the following conclusions:

1) On n-gram overlap metrics, the performance of large models does not meet our expectations, primarily because comparing the answers given by models like ChatGPT or other large code-oriented models with the ground truth on these metrics is not convincing. Their outputs often differ from the ground truth but are similarly informative.
2) In the future, we need more flexible objective metrics to evaluate the effectiveness of text generation tasks, especially involving LLMs.

D. Human Evaluation

We further carried out human evaluation experiments to assess the quality of the generation results. The setup of the human evaluation experiments strictly followed the experimental setup for human evaluation outlined in the main text.

TABLE A.III
HUMAN EVALUATION

Model	Informativeness	Naturalness	Similarity
APIMsGSum	2.016	2.640	2.524
Code Llama	1.109	3.213	2.892
GPT-3.5-turbo	1.210	3.311	2.927

Table A.III presents the results of our method and two types of LLMs in the human evaluation. From the table, we can observe:

1) Both types of large models score low on the Informativeness metric. This is because LLMs tend to generate summaries based on their own understanding, meaning these summaries are more influenced by the training set of the LLMs themselves, thereby creating a certain gap from the ground truth of the code summaries test set.
2) The results of LLMs score very high on the Naturalness metric. This is largely because LLMs are pre-trained on massive text datasets, enabling them to generate more fluent expressions.
3) The summaries generated by LLMs score higher on Similarity and surpass our method. This confirms our conjecture that the results of LLMs are also worth referring to, but it is inappropriate to assess them using n-gram overlap metrics. On the other hand, considering the significant differences between deep learning models and LLMs in terms of parameter size, computational resource consumption, and inference speed, there is no such a large gap in performance. We believe that, at the current stage, both models demonstrate their unique research and application values.

E. Conclusion

Overall, LLMs tend to have a larger scale of parameters and longer inference time, and using existing LLMs also incurs costs, such as OpenAI's model usage being explicitly priced. Therefore, they are more suitable as programming assistants for inquiries. By contrast, those based on deep learning have lower training costs and smaller parameter scales, making them easier to deploy. They can be directly used as IDE plugins for real-time code annotation generation. In different application scenarios, both types of models have their places of utility.

In comparison with pre-trained models of LLMs of the same parameter magnitude in the main text, our method achieves better performance with fewer parameters. Moreover, compared to LLMs that far exceed our model in terms of parameter quantity, we maintain a comparable level of effectiveness while benefiting from lower deployment costs and faster inference speed. On the other hand, one of the important contributions of our method, presented in our paper, is the idea and its implementation based on getting more complete code representation through API information. This can be slightly modified and used together with LLMs to achieve better effects.