SEMANTIC PREFIXES AS AN AUXILIARY TARGET FOR CODE UNDERSTANDING AND GENERATION

Kechen Qin[1], **Fan Yang**[1], **Kai-Wei Chang**[1,2], **Prashan Wanigasekara**[1], **Xinyue Liu**[3]
Chengwei Su[1], **Emre Barut**[1]
[1]Amazon Alexa AI, [2]University of California, Los Angeles, [3]Wish
qinkeche@amazon.com

ABSTRACT

Code understanding and generation require learning the mapping between human and programming languages. As human and programming languages are different in vocabulary, semantic, and, syntax, it is challenging for an autoregressive model to generate a sequence of tokens that is both semantically (i.e., carry the right meaning) and syntactically correct (i.e., in the right sequence order). Inspired by this, we propose a prefix-based learning framework to lessen the burden of an autoregressive generation model by decoupling the learning of semantic and syntactic dependencies. In particular, during the training we prepend the target output with a semantic embedding that encodes the output sequence. In this way, a model learns to first predict the semantics of the output before generating the sequence. Evaluating on 11 code understanding and generation datasets, we show that our prefix-prepending approach improves the baseline by an average of 8.1% in exact match and 5.5% in CodeBLEU. It also either outperforms or is on-par with the state-of-the-art methods across a variety of code understanding tasks. Our approach is general and can be used as a meta algorithm on top of any autoregressive language model.

1 INTRODUCTION

Pre-trained models such as GPT Brown et al. (2020) and BART Lewis et al. (2020) have empowered various natural language processing applications and brought them closer to a level that is practically applicable to real-world tasks. In the domain of software development, such models bring brand-new developer experiences: code generation conditioned on natural language prompts can save thousands of engineering hours; code translation methods can effectively scale current solutions and improve interoperability; and code summarization methods can be used for educational purposes, for managing large code bases and a plethora of other tasks.

While the state-of-the-art methods have shown their potential in understanding programming languages, they still have difficulty to provide production-level code. A recent study Mukherjee et al. (2021) shows that state-of-the-art models, such as Codex Chen et al. (2021a), cannot generate few lines of code that successfully pass code tests more than 50% of time. This induces the need to better understand and model the semantic gaps between human and programming languages. As shown in the example in Figure 1, code text has variable and function names (e.g., *"tol"* and *"abs"*) that are out of the context of the whole code block and the natural language prompt, which makes generation of such tokens especially difficult.

Recent state-of-the-art methods Chen et al. (2021b); Mukherjee et al. (2021) leverage a two-stage generation process to remedy the issue by first predicting semantic attributes in the target code (e.g., type of value of an argument) and then generating the full code conditioned on the output of the first stage. The hierarchical approach helps the model break down the problem into manageable components, but it breaks the end-to-end behavior and requires designing a task dependent two-step procedure.

To naturally ingest semantic information into an end-to-end code generation framework, we introduce a semantic prefix that contains compressed encoding of the target code. Using the prefix, we

achieve the aforementioned two-stage prediction in a single run of an encoder-decoder model by re-designing the model to predict the prefix as the first token in the sequence, along with the target. In our implementation, the semantic prefix is a high-level target code embedding, i.e., a summary representation, prepended to the target sequence. During inference, due to the autoregressive structure of the decoder, the model generates the prefix token first and predicts future tokens conditional on the generated prefix.

We also show that by leveraging a clustering model to group prefix tokens, we can increase the model accuracy through encouraging predictions of the model to be consistent on neighboring examples, Miyato et al. (2019); Wei et al. (2021).

In summary, we make the following contributions:

- We propose a novel prefix conditioning framework that leverages summary representations in code generation, translation, and summarization tasks. The prefix acts as an intermediary prediction step, that forces the model to approach the generation problem hierarchically. Particularly, our technique is not limited to the domain of code understanding, and can be applied to any type of autoregressive generation model.
- In order to improve the reliability of our approach, we build a clustering-based mechanism that forces consistency regularization on the model, and reduces the number of trainable parameters. Our technique only requires a minor change to the training procedure; it can be implemented with a few lines of code to any decoder based solution, and has no significant impact on training or inference latency.
- Experimental results show that our model improves the state-of-the-art results by an average of 1.42 CodeBLEU on three code generation datasets and by an average of 0.28 CodeBLEU on two code translation datasets. Further, the model achieves the best performance when summarizing JavaScript and Go, and is competitive for Ruby, Python and Java. Finally, when compared to baselines that has an identical architecture as ours, sans the prefix, our model significantly outperforms the baseline in all tasks by an average 8.1%, 5.6%, and 5.5% for EM, BLEU, and CodeBLEU scores, respectively.

2 Related Work

Code generation. Automatically generating code and their comments has been attractive area for research in the last few years. Earlier efforts focus on code generation and rely on rules to ensure compilability Aho & Johnson (1976); Aho et al. (1989); Domínguez et al. (2012). More recently, optimization-based models have achieved remarkable success to generate code by leveraging abstract syntax tree (AST) Maddison & Tarlow (2014); Yin & Neubig (2017); Murali et al. (2018); Alon et al. (2020); Peng et al. (2021); Kim et al. (2021); Mukherjee et al. (2021); Guo et al. (2021). Another line of works assume that models would learn the structure from large open-source code repositories and therefore relax modeling the AST. These works treat code as natural language and directly adopt language models to encode and generate code Clement et al. (2020); Ahmad et al. (2020); Svyatkovskiy et al. (2020); Perez et al. (2021); Shi et al. (2021); Parvez et al. (2021). Recently, billion-parameter transformer models such as Codex Chen et al. (2021a) and AlphaCode Li et al. (2022) report code generation applications for game building and competitive programming.

Prefix modeling for language generation. Sequence-to-sequence (Seq2seq) models Sutskever et al. (2014) have shown a great potential in language generation tasks, and can be enhanced by incorporating keywords extraction to generate informative outputs and avoid generic responses Hua & Wang (2018); Cho et al. (2019); Li et al. (2020); Wang et al. (2020); Hua & Wang (2020); Narayan et al. (2021). Shen et al. (2019) propose an E-M framework that predicts a multinomial latent variable before generating the target sequence, which can be seen as an early exploration of utilizing a prefix during the prediction process. In their work, the latent variable is mainly used for generating diverse hypotheses rather than connecting the input and the output. On a different track, various works improve Seq2seq models with the help of textual prompts Brown et al. (2020); Raffel et al. (2020); Sun & Lai (2020); Schick & Schütze (2021) by attaching task-specific prompts to input sequences. Prompt-based learning is mainly designed to be a lightweight alternative to fine-tuning, so in training only a small part of parameters are updated. The most relevant prompting research to ours are Ben-David et al. (2021) and Gu et al. (2021) who construct context-aware prompt, and Li & Liang

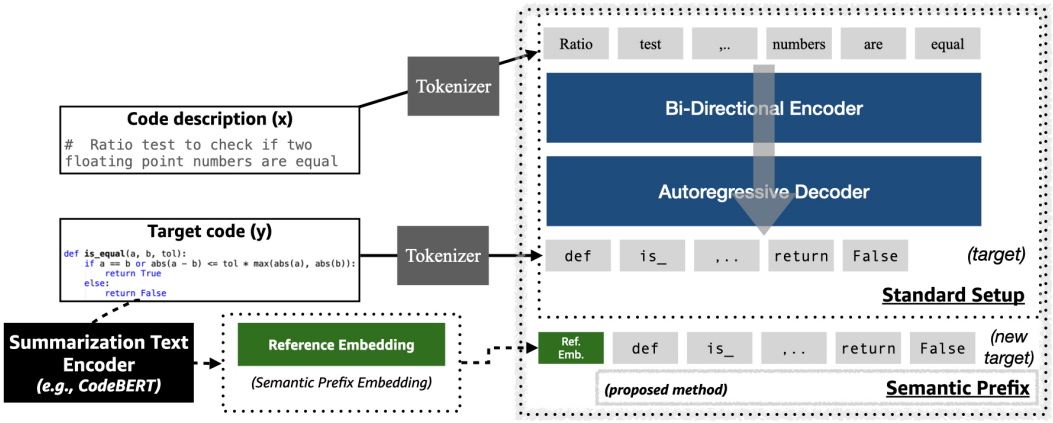

Figure 1: We demonstrate a standard encoder-decoder structure for code generation along with our proposed solution that leverages the semantic prefix, where the generated summary representation is prepended to the target code. The prefix embedding is obtained from the target sequence in training and predicted by the model during inference.

(2021) who propose to prepend prompt to the decoder. To the best of our knowledge, our work is the first one to utilize prefix augmentation in program understanding tasks.

3 Model

We consider a conditional generation task with two sequences of tokens, input context x and the target y. We aim to model $p(y|x)$ in code understanding scenarios, which relates to three tasks: text-to-code generation, code-to-text generation, and code-to-code translation.

3.1 Generation via Encoder-Decoder Architecture

A great number of language generation models adopt the encoder-decoder architecture Sutskever et al. (2014) to model $p(y|x)$, where x is encoded by the bidirectional Transformer Vaswani et al. (2017), and the decoder predicts y autoregressively conditioned on the encoded x and its left context. Following this track, we use BART Lewis et al. (2020) as the backbone structure in our main experiments. During pre-training, the model learns to recover the input text that is corrupted by some noising function, which is typically referred as denoising pre-training and shown to be quite effective in a broad set of generation tasks. We set corruption rate to be 35%, and use three noising strategies: token masking, token deletion, and token infilling. Specifically, we mask or delete a random token in masking and deletion, and replace a span of text with a single mask token in infilling. After pre-training on large-scale unlabeled programming language dataset containing various functions, we fine-tune the model on code and text pairs. Both pre-training and fine-tuning objectives can be written as $\max_\theta \sum_{n=1}^{N} \log p_\theta(y_n|x_n)$, where θ contains the model parameters and (x_n, y_n) is the n-th sample among the N many samples in the dataset. In the rest of the text, we use $p(\cdot)$ to denote $p_\theta(\cdot)$ when there is no ambiguity.

3.2 Forcing Hierarchical Thinking Through Prefix Prepending

Recent work in code generation has shown that two-step methods, that address the problem hierarchically by building macro level blocks (e.g., function declarations) first and details of the blocks (e.g., code for the function) later, significantly increase model accuracy – in some cases by over-performing models that are two orders of magnitude larger Mukherjee et al. (2021).

We propose to approximate the same procedure in our encoder-decoder setting by generating a summary representation, s_n, before predicting the target tokens y_n. The summary representation provides semantic context to the other tokens during the rest of the generation: The standard autoregressive decoder only makes prediction based on one side of the context, and is limited in its understanding of

Algorithm 1: MLE training for semantic prefix modeling

Input : Dataset $(x_n, y_n), n = 1, 2, ..., N$,
A separate pre-trained text encoder, ENCODER (e.g., CodeBERT)
Output : Trained model parameters

Obtain embeddings for the target sequence: $\tilde{s}_n = \text{ENCODER}(y_n)$ for $n = 1, \ldots, N$.
Use K-means on $\tilde{s}_n$ to obtain centroid representations for $\tilde{s}_n$, i.e., $h_n = \sum_{k=c_n} \bar{s}_k$ where $\bar{s}_k$ is the centroid for cluster k and c_n is the cluster id for $\tilde{s}_n$: $H = \text{KMEANS}(\tilde{s}_1, \ldots, \tilde{s}_N)$.
Prepend semantic prefix to the target y_n: $z_n = [s_n, y_n]$.
Train the model by maximizing: $\max_\theta \sum_n \log p(z_n|x_n)$.

Algorithm 2: EM training for semantic prefix modeling

Input : Inputs and Models of Algorithm 1
Output : Trained model parameters

foreach *epoch* **do**
    E-step: Obtain $\tilde{s}_{n,\text{epoch}}$ by running a forward pass and perform clustering:
    $\tilde{s}_{n,\text{epoch}} = forward(x_n)$
    $H_{\text{epoch}} = \text{KMEANS}(\tilde{s}_{1,\text{epoch}}, \ldots)$.
    Prepend semantic prefix to the targets: $z_{n,\text{epoch}} = [s_{n,\text{epoch}}, y_n]$.
    M-step: Optimize over θ to maximize $\sum_n \log p_\theta(z_{n,\text{epoch}}|x_n)$.
end

future context. Adding semantic prefix ensures that, at each time step the decoder accesses not only the previous tokens but also crucial information about relevant future tokens. This approach helps the model breakdown the problem into two manageable components. Accordingly, we write down the new training objective: $\max_\theta \sum_{n=1}^{N} \log p([s_n; y_n]|x_n) = \max_\theta \sum_{n=1}^{N} \log p(y_n|x_n, s_n) + \log p(s_n|x_n)$.

3.3 PREPENDING SEMANTIC PREFIX TO TARGET SEQUENCE

We employ a separate encoder (built for code and text summarization) to produce semantic prefix – a vector representation of the target sequence. The single prefix token captures the high-level information in the target sequence.

However, it is not reasonable to expect the decoder to be able to correctly predict the semantic prefix, as this would effectively require accurately solving both the code generation and the code summarization problem in the decoder. Setting more reasonable expectations on the model, we use an approximate proxy for the prefix embedding, where we cluster the summary representations of training samples via K-means, and use the cluster centroids as the semantic prefix. Overall, each target sequence is thus linked to a cluster, and the corresponding cluster centroid embedding is prepended to the target sequence in the embedding space. We train the model by maximizing the log-likelihood over the new (prefixed) target sequence. The procedure increases model consistency as samples in the same cluster are now forced to have the same prediction in the first generated token – a property known to reduce model generalization error Wei et al. (2021). The learning procedure is provided in Algorithm 1.

In the above training process, we allow the model to train and update the prefix parameters, but the cluster assignment for each sample stays the same, which results in additional variance in training. More precisely, the cluster assignments are determined before training (based on the initialization of the semantic prefixes), but the cluster assignments may change after the prefix embeddings are updated during training. To fix this issue, we design an EM-style training algorithm Dempster et al. (1977) to update cluster assignments after each epoch of training. Details are provided in Algorithm 2. Between each epoch, we update the prefix embeddings for each sample, and re-run the clustering algorithm to obtain new cluster assignments and centroids.

The combination of Algorithm 1 and Algorithm 2 composes our final solution. We find that running Algorithm 1 for several epochs without the EM step as a warm-up is effective at avoiding overfitting and local minima. Although generating prefix requires additional compute, the cost is minor in training. The additional steps require either filtering terms by regular expressions or running K-means.

Methods	EM	BLEU	CodeBLEU
Seq2Seq	3.05	21.31	26.39
Guo et al. (2019)	10.05	24.40	29.46
Iyer et al. (2019)	12.20	26.60	-
GPT-2	17.35	25.37	26.69
CodeGPT-2	18.25	28.69	32.71
CodeGPT-adapted	20.10	32.79	25.98
PLBART	18.75	36.69	38.52
CodeT5-small	21.55	38.13	41.49
CodeT5-base	22.30	**40.73**	43.20
UniXcoder w/o comment	21.45	37.15	-
UniXcoder	**22.60**	38.23	-
Ours	21.97	38.32	**44.21**

Table 1: Results of code generation task on CONCODE dataset.

In our experiments, the proposed method only runs 10% slower than the baseline model that has an identical architecture as ours, sans the prefix. The number could be further reduced by using more efficient K-means solutions.

Inference. During inference, our model works in the same manner as a standard autoregressive language model. The decoder generates the target sequence step by step, by first predicting a summary token and then target code tokens. This differs from the step in training, where the prefix needs to be obtained through regular expression matches or by using a separate text encoder and running K-means.

4 Experiments

4.1 Experimental Setup

We work with the CONCODE Iyer et al. (2018) dataset to evaluate our model on the text-to-code task (i.e., code generation). For the code-to-text task (i.e., code summarization) and code-to-code task (i.e., code translation) we use the CodeXGLUE Lu et al. (2021) dataset. Statistics about the datasets can be found in the Appendix A. In all experiments, we adopt exact match (EM), BLEU Papineni et al. (2002), and CodeBLEU Ren et al. (2020) as our evaluation metrics. Different from BLEU, CodeBLEU considers grammatical and logical correctness based on the abstract syntax tree and the data-flow structure; therefore we regard EM and CodeBLEU as the main evaluation metrics in tasks where the target is programming language, and report BLEU to make the reported numbers comparable with the existing literature. We compare our model with state-of-the-art models and obtain metrics for them from the respective papers. In the main experiments, we build our prefix component on top of the official PLBART implementation Ahmad et al. (2021) by using the Fairseq library Ott et al. (2019) (see Appendix C for detailed model structure). By comparing our model with the PLBART, we demonstrate the performance gain on various tasks by adding the semantic prefix. To build prefix, we employ CodeBERT Feng et al. (2020) to obtain summary representations of the targets and K-means Lloyd (1982) to perform clustering. Number of clusters is set to be 8 based on the elbow method Thorndike (1953). In evaluation, we remove the predicted prefix tokens and only evaluate on the predicted code sequence.

4.2 Code Generation

Table 1 shows the results for the code generation experiment on the CONCODE dataset. Our model achieves the best CodeBLUE score. Specifically, by increasing the model parameter count only by 0.004%, our model outperforms PLBART by 17.1% in terms of EM, 4.4% in terms of BLEU, and 14.7% in terms of CodeBLEU. This demonstrates the performance gain by adding the semantic prefix, as our model and PLBART share the same model architecture and training set except the usage of prefix.

Methods	Ruby	Javascript	Go	Python	Java	PHP
Seq2Seq	9.64	10.21	13.98	15.93	15.09	21.08
Transformer	11.18	11.59	16.38	15.81	16.26	22.12
RoBERTa	11.17	11.90	17.72	18.14	16.47	24.02
CodeBERT	12.16	14.90	18.07	19.06	17.65	25.16
PLBART	14.11	15.56	18.91	19.30	18.45	23.58
CodeT5-small	14.87	15.32	19.25	**20.04**	19.92	25.46
CodeT5-base	**15.24**	16.16	19.56	20.01	20.31	26.03
UniXcoder w/o comment	14.25	15.50	18.80	18.83	**20.25**	**26.17**
UniXcoder	14.87	15.85	19.07	19.13	20.31	**26.54**
Ours	14.54	**16.74**	**19.61**	19.43	19.50	24.07

Table 2: Results of source code summarization on CodeXGLUE dataset, evaluated with smoothed BLEU-4 score.

Methods	Java to C#			C# to Java		
	BLEU	EM	CodeBLEU	BLEU	EM	CodeBLEU
Naive Copy	18.54	0	34.20	18.69	0	43.04
PBSMT	43.53	12.50	42.71	40.06	16.10	43.48
Transformer	55.84	33.00	63.74	50.47	37.90	61.59
RoBERTa (code)	77.46	56.10	83.07	71.99	57.90	80.18
CodeBERT	79.92	59.00	85.10	72.14	58.80	79.41
GraphCodeBERT	80.58	59.40	-	72.64	58.80	-
PLBART	82.75	64.60	86.60	78.27	63.90	83.61
CodeT5-small	82.98	64.10	-	79.10	65.60	-
CodeT5-base	**84.03**	**65.90**	-	**79.87**	**66.90**	-
Ours	82.54	65.10	**86.91**	78.25	65.70	**83.86**

Table 3: Results of code translation task on CodeXGLUE dataset. Note that the PLBART results are slightly different from what Ahmad et al. reported in the paper due to using different checkpoints.

Our model performs slightly worse than CodeT5-base Wang et al. (2021) in terms of EM and BLEU and UniXcoder Guo et al. (2022) in terms of EM. The main reasons may be two-fold. Firstly, the gap may come from different pre-training datasets. UniXcoder includes 2.3M functions paired with comments from CodeSearchNet dataset Husain et al. (2019) in pre-training. PLBART shows that adding CodeSearchNet data to the pre-training stage improves the model performance by 4% on Python code generation task, so we argue that the gap between our model and the UniXcoder can be filled by incorporating the additional training data. Secondly, the gap is related to the model sizes. CodeT5-base uses a 12-layer encoder-decoder model, whose model size is 1.57 times larger than our model (220M vs. 140M). Likewise, when we compare with CodeT5-small (60M), a smaller version of CodeT5, our model outperforms it on all evaluation metrics.

4.3 Code Summarization

Different from code generation, summarization is a code-to-text task, which targets at translating source code to natural language. We benchmark our performance against PLBART and the state-of-the-art models in six programming languages on CodeXGLUE code-to-text dataset in Table 2. Our model outperforms all the comparison methods in JavaScript and Go. When comparing against PLBART, our model improves the result by an average of 0.66 BLEU. We also observe that our model fails to further improve on PHP language. Following previous work Ahmad et al. (2021) we hypothesize that there is a mismatch between the pre-trained languages (i.e., Python and Java) and PHP, and therefore it restricts the performance. Excluding PHP, our model achieves the second-best average BLEU over the rest 5 programming languages, and is only worse than CodeT5-base due to the difference between model sizes.

4.4 Code Translation

Table 3 presents the results of code translation task on CodeXGLUE code-to-code dataset. Our model outperforms all the comparison methods in CodeBLEU and remains the second best in EM due to different model size between our model and CodeT5-base. We highlight that our model performs

better than PLBART on all three generation tasks discussed in this section by an average 8.1%, 5.6% and 5.5% for EM, BLEU and CodeBLEU scores, respectively.

5 ANALYSIS

5.1 ABLATION STUDY

In order to breakdown the contributions of each factors in our solution, we conduct an ablation study to evaluate the performance of different training strategies in Table 4. We observe that the model without updating cluster assignment in training (i.e., removing Algorithm 2 from training) can still outperform the PLBART model, implying that the performance boost does not purely come from EM-style training. Further, in a complementary experiment, we see that the performance drops when the model starts without a warm-up stage (i.e., removing Algorithm 1 from training). In these cases, model becomes unsteady and gets stuck at local optima at the early stages of training, a common issue with EM-style methods Murphy (2012). Model instability becomes even more severe when the model is not initialized with a pre-trained model (e.g., CodeBERT). Finally, we observe significant performance changes when the number of clusters are varied. We experiment with no clustering, and with using 18 clusters (the number corresponds to the second-best elbow point). We see that after increasing the number of clusters beyond a certain point, the model starts to overfit and fails at generalization.

	Deletion from Algorithm	CodeBLEU
Ours (semantic prefix)	-	44.21
W/O updating prefix	Algorithm 2	-1.34
W/O warmup	Algorithm 1	-1.54
W/O CodeBERT	Line 1 in Algo 1	-1.77
Number of clusters=18	-	-1.17
W/O clustering	Line 2 in Algo 1 & 4 in Algo 2	-2.55

Table 4: Ablation study results on CONCODE dataset. We present variations of our different training algorithm by deleting certain lines of code from the Algorithm 1 and 2.

5.2 PLUGING PREFIX INTO VARIOUS ARCHITECTURES

In our previous experiments, we use BART as the base model and demonstrate performance gains on various tasks by adding the prefix. In this section, we experiment with integrating our idea into different architectures to show that our method can adapt to any type of autoregressive generation model. We consider three code understanding models, CodeBERT, PLBART, and CodeT5, that leverage three different architectures: RoBERTa$_{base}$ Liu et al. (2019), BART$_{base}$ Lewis et al. (2020), and T5$_{base}$ Raffel et al. (2020). Adding a semantic prefix increases the model parameter count by 0.0035%, 0.0040%, and 0.0027% for the 3 models.

We present the experimental results in Table 5. The models which are equipped with the prefix perform better than the vanilla versions on all tasks, except for the code-to-code task with PLBART. The improvements are more significant on text-to-code task than the other two tasks. This observation confirms the advantages of modeling a semantic prefix in a task where the input and output are in different languages. Notably, our method is more beneficial when the prediction target is a programming language. We hypothesize that the prefix provides summarization of both the semantic and structured information of the target program. The semantic information is captured by encoding using the summarization encoder, and the structured information is acquired through clustering.

5.3 COMPARISON TO PLANNING-BASED GENERATION WITH DISCRETE PREFIX

As discussed in previous work Hua & Wang (2020); Narayan et al. (2021), keywords in target can be modeled as prefix to provide intermediate plan to ground the generation. This so-called planning-based generation model learns to predict a summary plan in the form of a chain of keywords, followed by the target sequence. Inspired by their work, in the code generation setting, we build a different type of prefix – a sequence of keywords from target program, such as function and variable names used in the function calls. That gives rise to the discrete prefix method. The discrete prefix

Model	Architecture	Task	BLEU			EM			CodeBLEU		
			w/o prefix	w/ prefix	relative change	w/o prefix	w/ prefix	relative change	w/o prefix	w/ prefix	relative change
CodeBERT	RoBERTa-base	text-to-code	33.32	34.99	1.67	19.35	20.75	1.40	36.48	38.14	1.66
		code-to-text	14.90	15.66	0.76	-	-	-	-	-	-
		code-to-code	79.92	80.71	0.78	59.00	61.43	2.43	85.10	85.32	0.22
PLBART	BART-base	text-to-code	36.69	38.32	1.63	18.75	21.97	3.22	38.52	44.21	5.69
		code-to-text	15.56	16.74	1.18	-	-	-	-	-	-
		code-to-code	82.75	82.54	-0.21	64.60	65.10	0.50	86.60	86.91	0.31
CodeT5	T5-base	text-to-code	40.73	40.78	0.05	22.30	22.55	0.25	43.20	44.13	0.93
		code-to-text	16.16	16.68	0.52	-	-	-	-	-	-
		code-to-code	84.03	84.45	0.42	65.90	67.0	1.10	88.04	88.36	0.32

Table 5: Relative improvement scores with prefix integration for 3 model architectures and 3 code/text generation tasks. For each task, we conduct experiment on one of the most representative datasets; we use CONCODE, CodeXGLUE (JavaScript), and CodeXGLUE (Java-to-C#) for text-to-code, code-to-text and code-to-code, respectively. In the code-to-text task, we report the Smoothed BLEU-4.

Methods	JuICe	CodeXGLUE	CONCODE
w/o prefix	41.61	14.39	38.52
w/ discrete prefix	41.67	21.12	41.36
w/ proposed prefix	43.23	21.78	44.21

Table 6: CodeBLEU of applying planning-based generation model (i.e., w/ discrete prefix) and our proposed method (i.e., w/ semantic prefix) on 3 code generation datasets.

approach enables us to control the content in the prefix by manually selecting the summary-worthy tokens and prepending them to the target sequence.

Specifically, our discrete prefix learning framework consists of three steps: (i) we use regular expression to extract function names and arguments from the target code; (ii) we prepend the target code y_n with the tokens for the extracted sequences, s_n, forming $z_n = [s_n, y_n]$; (iii) we train the model over the new target z_n. Take the prompt-code pair in Figure 1 as an example – we obtain three discrete tokens *"is_equal"*, *"tol"*, and *"abs"* which capture high-level details of the target code whose purpose is to *check equality of two floating numbers*. If the decoder successfully generates the prefix, it acts as an effective guide as the model knows that it should be using the *"abs"* method and the *"tol"* variable.

To compare discrete prefix with our proposed semantic prefix, we test both methods on 3 code generation datasets, JuICe Agashe et al. (2019), text-to-python dataset in CodeXGLUE, and CONCODE. We observe that both discrete prefix and semantic prefix significantly outperform the baseline that has an identical architecture, sans the prefix. This trend is consistent across all three datasets, which confirms the advantage of modeling summary prefix, despite making prefix discrete or continuous. Our proposed method is more advantageous than discrete prefix. The reasons may be two-fold. Firstly, experimental results (see Appendix D) show that different selections of summary tokens have a strong impact on the model performance. It is challenging to find the optimal discrete prefix, while the semantic prefix is trained in training. Secondly, discrete tokens have less capacity and flexibility to summarize training target. They are limited to be tokens extracted from the target sequence, and their representations are limited to be in the decoder's token space. We conclude that it is not necessary to limit the prefix to human-interpretable natural language, as our goal is to have the model perform the generation task with the help of prefix.

6 Conclusion

We propose a method to improve model accuracy in code understanding and generation tasks by adding a semantic prefix to the target. During the training we prepend the target output with a semantic embedding that encodes the output sequence. In this way, a model learns to first predict the semantics of the output before generating the sequence. Evaluating on 11 code understanding and generation datasets, we show that out prefix-prepending approach improves the baseline by an average of 8.1% in exact match and 5.5% in CodeBLEU. It also either outperforms or is on-par with the state-of-the-art methods across a variety of code understanding tasks. Our approach is general and can be use as a meta algorithm on top of any autoregressive language model.

REFERENCES

Rajas Agashe, Srinivasan Iyer, and Luke Zettlemoyer. Juice: A large scale distantly supervised dataset for open domain context-based code generation. In Kentaro Inui, Jing Jiang, Vincent Ng, and Xiaojun Wan (eds.), *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing, EMNLP-IJCNLP 2019, Hong Kong, China, November 3-7, 2019*, pp. 5435–5445. Association for Computational Linguistics, 2019. doi: 10.18653/v1/D19-1546. URL https://doi.org/10.18653/v1/D19-1546.

Wasi Uddin Ahmad, Saikat Chakraborty, Baishakhi Ray, and Kai-Wei Chang. A transformer-based approach for source code summarization. In Dan Jurafsky, Joyce Chai, Natalie Schluter, and Joel R. Tetreault (eds.), *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics, ACL 2020, Online, July 5-10, 2020*, pp. 4998–5007. Association for Computational Linguistics, 2020. doi: 10.18653/v1/2020.acl-main.449. URL https://doi.org/10.18653/v1/2020.acl-main.449.

Wasi Uddin Ahmad, Saikat Chakraborty, Baishakhi Ray, and Kai-Wei Chang. Unified pre-training for program understanding and generation. In Kristina Toutanova, Anna Rumshisky, Luke Zettlemoyer, Dilek Hakkani-Tür, Iz Beltagy, Steven Bethard, Ryan Cotterell, Tanmoy Chakraborty, and Yichao Zhou (eds.), *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HLT 2021, Online, June 6-11, 2021*, pp. 2655–2668. Association for Computational Linguistics, 2021. doi: 10.18653/v1/2021.naacl-main.211. URL https://doi.org/10.18653/v1/2021.naacl-main.211.

Alfred V. Aho and Stephen C. Johnson. Optimal code generation for expression trees. *J. ACM*, 23 (3):488–501, 1976. doi: 10.1145/321958.321970. URL https://doi.org/10.1145/321958.321970.

Alfred V. Aho, Mahadevan Ganapathi, and Steven W. K. Tjiang. Code generation using tree matching and dynamic programming. *ACM Trans. Program. Lang. Syst.*, 11(4):491–516, 1989. doi: 10.1145/69558.75700. URL https://doi.org/10.1145/69558.75700.

Uri Alon, Roy Sadaka, Omer Levy, and Eran Yahav. Structural language models of code. In *Proceedings of the 37th International Conference on Machine Learning, ICML 2020, 13-18 July 2020, Virtual Event*, volume 119 of *Proceedings of Machine Learning Research*, pp. 245–256. PMLR, 2020. URL http://proceedings.mlr.press/v119/alon20a.html.

Eyal Ben-David, Nadav Oved, and Roi Reichart. PADA: A prompt-based autoregressive approach for adaptation to unseen domains. *CoRR*, abs/2102.12206, 2021. URL https://arxiv.org/abs/2102.12206.

Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. Language models are few-shot learners. In Hugo Larochelle, Marc'Aurelio Ranzato, Raia Hadsell, Maria-Florina Balcan, and Hsuan-Tien Lin (eds.), *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*, 2020. URL https://proceedings.neurips.cc/paper/2020/hash/1457c0d6bfcb4967418bfb8ac142f64a-Abstract.html.

Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, Jared Kaplan, Harrison Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, Alex Ray, Raul Puri, Gretchen Krueger, Michael Petrov, Heidy Khlaaf, Girish Sastry, Pamela Mishkin, Brooke Chan, Scott Gray, Nick Ryder, Mikhail Pavlov, Alethea Power, Lukasz Kaiser, Mohammad Bavarian, Clemens Winter, Philippe Tillet, Felipe Petroski Such, Dave Cummings, Matthias Plappert, Fotios Chantzis, Elizabeth Barnes, Ariel Herbert-Voss, William Hebgen Guss, Alex Nichol, Alex Paino, Nikolas Tezak, Jie Tang, Igor Babuschkin, Suchir Balaji, Shantanu Jain, William Saunders, Christopher Hesse, Andrew N. Carr, Jan Leike, Joshua Achiam, Vedant Misra, Evan Morikawa,

Alec Radford, Matthew Knight, Miles Brundage, Mira Murati, Katie Mayer, Peter Welinder, Bob McGrew, Dario Amodei, Sam McCandlish, Ilya Sutskever, and Wojciech Zaremba. Evaluating large language models trained on code. *CoRR*, abs/2107.03374, 2021a. URL https://arxiv.org/abs/2107.03374.

Xinyun Chen, Linyuan Gong, Alvin Cheung, and Dawn Song. Plotcoder: Hierarchical decoding for synthesizing visualization code in programmatic context. In Chengqing Zong, Fei Xia, Wenjie Li, and Roberto Navigli (eds.), *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing, ACL/IJCNLP 2021, (Volume 1: Long Papers), Virtual Event, August 1-6, 2021*, pp. 2169–2181. Association for Computational Linguistics, 2021b. doi: 10.18653/v1/2021.acl-long.169. URL https://doi.org/10.18653/v1/2021.acl-long.169.

Jaemin Cho, Min Joon Seo, and Hannaneh Hajishirzi. Mixture content selection for diverse sequence generation. In Kentaro Inui, Jing Jiang, Vincent Ng, and Xiaojun Wan (eds.), *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing, EMNLP-IJCNLP 2019, Hong Kong, China, November 3-7, 2019*, pp. 3119–3129. Association for Computational Linguistics, 2019. doi: 10.18653/v1/D19-1308. URL https://doi.org/10.18653/v1/D19-1308.

Colin B. Clement, Dawn Drain, Jonathan Timcheck, Alexey Svyatkovskiy, and Neel Sundaresan. Pymt5: multi-mode translation of natural language and python code with transformers. In Bonnie Webber, Trevor Cohn, Yulan He, and Yang Liu (eds.), *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing, EMNLP 2020, Online, November 16-20, 2020*, pp. 9052–9065. Association for Computational Linguistics, 2020. doi: 10.18653/v1/2020.emnlp-main.728. URL https://doi.org/10.18653/v1/2020.emnlp-main.728.

A. P. Dempster, N. M. Laird, and D. B. Rubin. Maximum likelihood from incomplete data via the EM algorithm. *Journal of the Royal Statistical Society: Series B*, 39:1–38, 1977. URL http://web.mit.edu/6.435/www/Dempster77.pdf.

Eladio Domínguez, Beatriz Pérez, Ángel Luis Rubio, and María A. Zapata. A systematic review of code generation proposals from state machine specifications. *Inf. Softw. Technol.*, 54(10):1045–1066, 2012. doi: 10.1016/j.infsof.2012.04.008. URL https://doi.org/10.1016/j.infsof.2012.04.008.

Zhangyin Feng, Daya Guo, Duyu Tang, Nan Duan, Xiaocheng Feng, Ming Gong, Linjun Shou, Bing Qin, Ting Liu, Daxin Jiang, and Ming Zhou. Codebert: A pre-trained model for programming and natural languages. In Trevor Cohn, Yulan He, and Yang Liu (eds.), *Findings of the Association for Computational Linguistics: EMNLP 2020, Online Event, 16-20 November 2020*, volume EMNLP 2020 of *Findings of ACL*, pp. 1536–1547. Association for Computational Linguistics, 2020. doi: 10.18653/v1/2020.findings-emnlp.139. URL https://doi.org/10.18653/v1/2020.findings-emnlp.139.

Xiaodong Gu, Kang Min Yoo, and Sang-Woo Lee. Response generation with context-aware prompt learning. *CoRR*, abs/2111.02643, 2021. URL https://arxiv.org/abs/2111.02643.

Daya Guo, Duyu Tang, Nan Duan, Ming Zhou, and Jian Yin. Coupling retrieval and meta-learning for context-dependent semantic parsing. In Anna Korhonen, David R. Traum, and Lluís Màrquez (eds.), *Proceedings of the 57th Conference of the Association for Computational Linguistics, ACL 2019, Florence, Italy, July 28- August 2, 2019, Volume 1: Long Papers*, pp. 855–866. Association for Computational Linguistics, 2019. doi: 10.18653/v1/p19-1082. URL https://doi.org/10.18653/v1/p19-1082.

Daya Guo, Shuo Ren, Shuai Lu, Zhangyin Feng, Duyu Tang, Shujie Liu, Long Zhou, Nan Duan, Alexey Svyatkovskiy, Shengyu Fu, Michele Tufano, Shao Kun Deng, Colin B. Clement, Dawn Drain, Neel Sundaresan, Jian Yin, Daxin Jiang, and Ming Zhou. Graphcodebert: Pre-training code representations with data flow. In *9th International Conference on Learning Representations, ICLR 2021, Virtual Event, Austria, May 3-7, 2021*. OpenReview.net, 2021. URL https://openreview.net/forum?id=jLoC4ez43PZ.

Daya Guo, Shuai Lu, Nan Duan, Yanlin Wang, Ming Zhou, and Jian Yin. Unixcoder: Unified cross-modal pre-training for code representation. *CoRR*, abs/2203.03850, 2022. doi: 10.48550/arXiv.2203.03850. URL https://doi.org/10.48550/arXiv.2203.03850.

Xinyu Hua and Lu Wang. Neural argument generation augmented with externally retrieved evidence. In Iryna Gurevych and Yusuke Miyao (eds.), *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics, ACL 2018, Melbourne, Australia, July 15-20, 2018, Volume 1: Long Papers*, pp. 219–230. Association for Computational Linguistics, 2018. doi: 10.18653/v1/P18-1021. URL https://aclanthology.org/P18-1021/.

Xinyu Hua and Lu Wang. PAIR: Planning and iterative refinement in pre-trained transformers for long text generation. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pp. 781–793, Online, November 2020. Association for Computational Linguistics. doi: 10.18653/v1/2020.emnlp-main.57. URL https://aclanthology.org/2020.emnlp-main.57.

Hamel Husain, Ho-Hsiang Wu, Tiferet Gazit, Miltiadis Allamanis, and Marc Brockschmidt. Codesearchnet challenge: Evaluating the state of semantic code search. *CoRR*, abs/1909.09436, 2019. URL http://arxiv.org/abs/1909.09436.

Srinivasan Iyer, Ioannis Konstas, Alvin Cheung, and Luke Zettlemoyer. Mapping language to code in programmatic context. In Ellen Riloff, David Chiang, Julia Hockenmaier, and Jun'ichi Tsujii (eds.), *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing, Brussels, Belgium, October 31 - November 4, 2018*, pp. 1643–1652. Association for Computational Linguistics, 2018. doi: 10.18653/v1/d18-1192. URL https://doi.org/10.18653/v1/d18-1192.

Srinivasan Iyer, Alvin Cheung, and Luke Zettlemoyer. Learning programmatic idioms for scalable semantic parsing. In Kentaro Inui, Jing Jiang, Vincent Ng, and Xiaojun Wan (eds.), *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing, EMNLP-IJCNLP 2019, Hong Kong, China, November 3-7, 2019*, pp. 5425–5434. Association for Computational Linguistics, 2019. doi: 10.18653/v1/D19-1545. URL https://doi.org/10.18653/v1/D19-1545.

Seohyun Kim, Jinman Zhao, Yuchi Tian, and Satish Chandra. Code prediction by feeding trees to transformers. In *43rd IEEE/ACM International Conference on Software Engineering, ICSE 2021, Madrid, Spain, 22-30 May 2021*, pp. 150–162. IEEE, 2021. doi: 10.1109/ICSE43902.2021.00026. URL https://doi.org/10.1109/ICSE43902.2021.00026.

Mike Lewis, Yinhan Liu, Naman Goyal, Marjan Ghazvininejad, Abdelrahman Mohamed, Omer Levy, Veselin Stoyanov, and Luke Zettlemoyer. BART: denoising sequence-to-sequence pre-training for natural language generation, translation, and comprehension. In Dan Jurafsky, Joyce Chai, Natalie Schluter, and Joel R. Tetreault (eds.), *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics, ACL 2020, Online, July 5-10, 2020*, pp. 7871–7880. Association for Computational Linguistics, 2020. doi: 10.18653/v1/2020.acl-main.703. URL https://doi.org/10.18653/v1/2020.acl-main.703.

Haoran Li, Junnan Zhu, Jiajun Zhang, Chengqing Zong, and Xiaodong He. Keywords-guided abstractive sentence summarization. In *The Thirty-Fourth AAAI Conference on Artificial Intelligence, AAAI 2020, The Thirty-Second Innovative Applications of Artificial Intelligence Conference, IAAI 2020, The Tenth AAAI Symposium on Educational Advances in Artificial Intelligence, EAAI 2020, New York, NY, USA, February 7-12, 2020*, pp. 8196–8203. AAAI Press, 2020. URL https://ojs.aaai.org/index.php/AAAI/article/view/6333.

Xiang Lisa Li and Percy Liang. Prefix-tuning: Optimizing continuous prompts for generation. In Chengqing Zong, Fei Xia, Wenjie Li, and Roberto Navigli (eds.), *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing, ACL/IJCNLP 2021, (Volume 1: Long Papers), Virtual Event, August 1-6, 2021*, pp. 4582–4597. Association for Computational Linguistics, 2021. doi: 10.18653/v1/2021.acl-long.353. URL https://doi.org/10.18653/v1/2021.acl-long.353.

Yujia Li, David H. Choi, Junyoung Chung, Nate Kushman, Julian Schrittwieser, Rémi Leblond, Tom Eccles, James Keeling, Felix Gimeno, Agustin Dal Lago, Thomas Hubert, Peter Choy, Cyprien de Masson d'Autume, Igor Babuschkin, Xinyun Chen, Po-Sen Huang, Johannes Welbl, Sven Gowal, Alexey Cherepanov, James Molloy, Daniel J. Mankowitz, Esme Sutherland Robson, Pushmeet Kohli, Nando de Freitas, Koray Kavukcuoglu, and Oriol Vinyals. Competition-level code generation with alphacode. *CoRR*, abs/2203.07814, 2022. doi: 10.48550/arXiv.2203.07814. URL https://doi.org/10.48550/arXiv.2203.07814.

Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. Roberta: A robustly optimized BERT pretraining approach. *CoRR*, abs/1907.11692, 2019. URL http://arxiv.org/abs/1907.11692.

Stuart P. Lloyd. Least squares quantization in PCM. *IEEE Trans. Inf. Theory*, 28(2):129–136, 1982. doi: 10.1109/TIT.1982.1056489. URL https://doi.org/10.1109/TIT.1982.1056489.

Shuai Lu, Daya Guo, Shuo Ren, Junjie Huang, Alexey Svyatkovskiy, Ambrosio Blanco, Colin B. Clement, Dawn Drain, Daxin Jiang, Duyu Tang, Ge Li, Lidong Zhou, Linjun Shou, Long Zhou, Michele Tufano, Ming Gong, Ming Zhou, Nan Duan, Neel Sundaresan, Shao Kun Deng, Shengyu Fu, and Shujie Liu. Codexglue: A machine learning benchmark dataset for code understanding and generation. In Joaquin Vanschoren and Sai-Kit Yeung (eds.), *Proceedings of the Neural Information Processing Systems Track on Datasets and Benchmarks 1, NeurIPS Datasets and Benchmarks 2021, December 2021, virtual*, 2021. URL https://datasets-benchmarks-proceedings.neurips.cc/paper/2021/hash/c16a5320fa475530d9583c34fd356ef5-Abstract-round1.html.

Chris J. Maddison and Daniel Tarlow. Structured generative models of natural source code. In *Proceedings of the 31th International Conference on Machine Learning, ICML 2014, Beijing, China, 21-26 June 2014*, volume 32 of *JMLR Workshop and Conference Proceedings*, pp. 649–657. JMLR.org, 2014. URL http://proceedings.mlr.press/v32/maddison14.html.

Takeru Miyato, Shin-ichi Maeda, Masanori Koyama, and Shin Ishii. Virtual adversarial training: A regularization method for supervised and semi-supervised learning. *IEEE Trans. Pattern Anal. Mach. Intell.*, 41(8):1979–1993, 2019. doi: 10.1109/TPAMI.2018.2858821. URL https://doi.org/10.1109/TPAMI.2018.2858821.

Rohan Mukherjee, Yeming Wen, Dipak Chaudhari, Thomas W. Reps, Swarat Chaudhuri, and Christopher M. Jermaine. Neural program generation modulo static analysis. In Marc'Aurelio Ranzato, Alina Beygelzimer, Yann N. Dauphin, Percy Liang, and Jennifer Wortman Vaughan (eds.), *Advances in Neural Information Processing Systems 34: Annual Conference on Neural Information Processing Systems 2021, NeurIPS 2021, December 6-14, 2021, virtual*, pp. 18984–18996, 2021. URL https://proceedings.neurips.cc/paper/2021/hash/9e1a36515d6704d7eb7a30d783400e5d-Abstract.html.

Vijayaraghavan Murali, Letao Qi, Swarat Chaudhuri, and Chris Jermaine. Neural sketch learning for conditional program generation. In *6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Conference Track Proceedings*. OpenReview.net, 2018. URL https://openreview.net/forum?id=HkfXMz-Ab.

Kevin P. Murphy. *Machine learning - a probabilistic perspective*. Adaptive computation and machine learning series. MIT Press, 2012. ISBN 0262018020.

Shashi Narayan, Yao Zhao, Joshua Maynez, Gonçalo Simões, Vitaly Nikolaev, and Ryan McDonald. Planning with learned entity prompts for abstractive summarization. *Transactions of the Association for Computational Linguistics*, 9:1475–1492, 2021. doi: 10.1162/tacl_a_00438. URL https://aclanthology.org/2021.tacl-1.88.

Myle Ott, Sergey Edunov, Alexei Baevski, Angela Fan, Sam Gross, Nathan Ng, David Grangier, and Michael Auli. fairseq: A fast, extensible toolkit for sequence modeling. In Waleed Ammar, Annie Louis, and Nasrin Mostafazadeh (eds.), *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HLT 2019, Minneapolis, MN, USA, June 2-7, 2019, Demonstrations*, pp. 48–53. Association for Computational Linguistics, 2019. doi: 10.18653/v1/n19-4009. URL https://doi.org/10.18653/v1/n19-4009.

Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. Bleu: a method for automatic evaluation of machine translation. In *Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics, July 6-12, 2002, Philadelphia, PA, USA*, pp. 311–318. ACL, 2002. doi: 10.3115/1073083.1073135. URL https://aclanthology.org/P02-1040/.

Md. Rizwan Parvez, Wasi Uddin Ahmad, Saikat Chakraborty, Baishakhi Ray, and Kai-Wei Chang. Retrieval augmented code generation and summarization. In Marie-Francine Moens, Xuanjing Huang, Lucia Specia, and Scott Wen-tau Yih (eds.), *Findings of the Association for Computational Linguistics: EMNLP 2021, Virtual Event / Punta Cana, Dominican Republic, 16-20 November, 2021*, pp. 2719–2734. Association for Computational Linguistics, 2021. doi: 10.18653/v1/2021.findings-emnlp.232. URL https://doi.org/10.18653/v1/2021.findings-emnlp.232.

Han Peng, Ge Li, Wenhan Wang, Yunfei Zhao, and Zhi Jin. Integrating tree path in transformer for code representation. In Marc'Aurelio Ranzato, Alina Beygelzimer, Yann N. Dauphin, Percy Liang, and Jennifer Wortman Vaughan (eds.), *Advances in Neural Information Processing Systems 34: Annual Conference on Neural Information Processing Systems 2021, NeurIPS 2021, December 6-14, 2021, virtual*, pp. 9343–9354, 2021. URL https://proceedings.neurips.cc/paper/2021/hash/4e0223a87610176ef0d24ef6d2dcde3a-Abstract.html.

Luis Perez, Lizi Ottens, and Sudharshan Viswanathan. Automatic code generation using pre-trained language models. *CoRR*, abs/2102.10535, 2021. URL https://arxiv.org/abs/2102.10535.

Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J. Liu. Exploring the limits of transfer learning with a unified text-to-text transformer. *J. Mach. Learn. Res.*, 21:140:1–140:67, 2020. URL http://jmlr.org/papers/v21/20-074.html.

Shuo Ren, Daya Guo, Shuai Lu, Long Zhou, Shujie Liu, Duyu Tang, Neel Sundaresan, Ming Zhou, Ambrosio Blanco, and Shuai Ma. Codebleu: a method for automatic evaluation of code synthesis. *CoRR*, abs/2009.10297, 2020. URL https://arxiv.org/abs/2009.10297.

Timo Schick and Hinrich Schütze. It's not just size that matters: Small language models are also few-shot learners. In Kristina Toutanova, Anna Rumshisky, Luke Zettlemoyer, Dilek Hakkani-Tür, Iz Beltagy, Steven Bethard, Ryan Cotterell, Tanmoy Chakraborty, and Yichao Zhou (eds.), *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HLT 2021, Online, June 6-11, 2021*, pp. 2339–2352. Association for Computational Linguistics, 2021. doi: 10.18653/v1/2021.naacl-main.185. URL https://doi.org/10.18653/v1/2021.naacl-main.185.

Tianxiao Shen, Myle Ott, Michael Auli, and Marc'Aurelio Ranzato. Mixture models for diverse machine translation: Tricks of the trade. In Kamalika Chaudhuri and Ruslan Salakhutdinov (eds.), *Proceedings of the 36th International Conference on Machine Learning, ICML 2019, 9-15 June 2019, Long Beach, California, USA*, volume 97 of *Proceedings of Machine Learning Research*, pp. 5719–5728. PMLR, 2019. URL http://proceedings.mlr.press/v97/shen19c.html.

Ensheng Shi, Yanlin Wang, Lun Du, Hongyu Zhang, Shi Han, Dongmei Zhang, and Hongbin Sun. CAST: enhancing code summarization with hierarchical splitting and reconstruction of abstract syntax trees. In Marie-Francine Moens, Xuanjing Huang, Lucia Specia, and Scott Wen-tau Yih (eds.), *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing, EMNLP 2021, Virtual Event / Punta Cana, Dominican Republic, 7-11 November, 2021*, pp. 4053–4062. Association for Computational Linguistics, 2021. doi: 10.18653/v1/2021.emnlp-main.332. URL https://doi.org/10.18653/v1/2021.emnlp-main.332.

Fan-Keng Sun and Cheng-I Lai. Conditioned natural language generation using only unconditioned language model: An exploration. *CoRR*, abs/2011.07347, 2020. URL https://arxiv.org/abs/2011.07347.

Ilya Sutskever, Oriol Vinyals, and Quoc V. Le. Sequence to sequence learning with neural networks. In Zoubin Ghahramani, Max Welling, Corinna Cortes, Neil D. Lawrence, and Kilian Q. Weinberger (eds.), *Advances in Neural Information Processing Systems 27: Annual Conference on Neural Information Processing Systems 2014, December 8-13 2014, Montreal, Quebec, Canada*, pp. 3104–3112, 2014. URL https://proceedings.neurips.cc/paper/2014/hash/a14ac55a4f27472c5d894ec1c3c743d2-Abstract.html.

Alexey Svyatkovskiy, Shao Kun Deng, Shengyu Fu, and Neel Sundaresan. Intellicode compose: code generation using transformer. In Prem Devanbu, Myra B. Cohen, and Thomas Zimmermann (eds.), *ESEC/FSE '20: 28th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering, Virtual Event, USA, November 8-13, 2020*, pp. 1433–1443. ACM, 2020. doi: 10.1145/3368089.3417058. URL https://doi.org/10.1145/3368089.3417058.

Robert Thorndike. Who belongs in the family? *Psychometrika*, 18(4):267–276, 1953. URL https://EconPapers.repec.org/RePEc:spr:psycho:v:18:y:1953:i:4:p:267-276.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. In Isabelle Guyon, Ulrike von Luxburg, Samy Bengio, Hanna M. Wallach, Rob Fergus, S. V. N. Vishwanathan, and Roman Garnett (eds.), *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, December 4-9, 2017, Long Beach, CA, USA*, pp. 5998–6008, 2017. URL https://proceedings.neurips.cc/paper/2017/hash/3f5ee243547dee91fbd053c1c4a845aa-Abstract.html.

Yue Wang, Weishi Wang, Shafiq R. Joty, and Steven C. H. Hoi. Codet5: Identifier-aware unified pre-trained encoder-decoder models for code understanding and generation. In Marie-Francine Moens, Xuanjing Huang, Lucia Specia, and Scott Wen-tau Yih (eds.), *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing, EMNLP 2021, Virtual Event / Punta Cana, Dominican Republic, 7-11 November, 2021*, pp. 8696–8708. Association for Computational Linguistics, 2021. doi: 10.18653/v1/2021.emnlp-main.685. URL https://doi.org/10.18653/v1/2021.emnlp-main.685.

Zhen Wang, Siwei Rao, Jie Zhang, Zhen Qin, Guangjian Tian, and Jun Wang. Diversify question generation with continuous content selectors and question type modeling. In Trevor Cohn, Yulan He, and Yang Liu (eds.), *Findings of the Association for Computational Linguistics: EMNLP 2020, Online Event, 16-20 November 2020*, volume EMNLP 2020 of *Findings of ACL*, pp. 2134–2143. Association for Computational Linguistics, 2020. doi: 10.18653/v1/2020.findings-emnlp.194. URL https://doi.org/10.18653/v1/2020.findings-emnlp.194.

Colin Wei, Kendrick Shen, Yining Chen, and Tengyu Ma. Theoretical analysis of self-training with deep networks on unlabeled data. In *9th International Conference on Learning Representations, ICLR 2021, Virtual Event, Austria, May 3-7, 2021*. OpenReview.net, 2021. URL https://openreview.net/forum?id=rC8sJ4i6kaH.

Pengcheng Yin and Graham Neubig. A syntactic neural model for general-purpose code generation. In Regina Barzilay and Min-Yen Kan (eds.), *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics, ACL 2017, Vancouver, Canada, July 30 - August 4, Volume 1: Long Papers*, pp. 440–450. Association for Computational Linguistics, 2017. doi: 10.18653/v1/P17-1041. URL https://doi.org/10.18653/v1/P17-1041.

A Data Statistics

We present data statistics in Table 7.

B Connections to Prompting and Prefix-Tuning

Prepending a prefix is related to soft-prompting, which appends additional embeddings to the input to improve model accuracy Li & Liang (2021). To be more precise, prompting appends context meaningful tokens to the input text, to form new inputs $[x; s]$. The method relies on the fact that the prompt lifts the scores of the more accurate outputs, i.e., over a high probability set of (x, y) pairs, its holds that $p(y|x, s) \geq p(y|x)$.

In our setting, the first term in the objective function, $\log p(y_n|x_n, s_n)$, optimizes the same likelihood as soft-prompting. There is one major difference: s_n in our setting represents a summary representation of the target and not input level deterministic tokens. The second term, $\log p(s_n|x_n)$, acts as

Task	Dataset	Language	Train	Valid	Test
Generation	JuICe	NL to Python	38,971	827	894
	CodeXGLUE	NL to Python	251,820	13,914	14,918
	CONCODE	NL to Java	100,000	2,000	2,000
Summarization	CodeXGLUE	Ruby	24,927	1,400	1,216
		JavaScript	58,025	3,885	3,291
		Go	167,288	7,325	8,122
		Python	251,820	13,914	14,918
		Java	164,923	5,183	10,955
		PHP	241,241	12,982	14,014
Translation	CodeXGLUE	Java to C#	10,300	500	1000
		C# to Java	10,300	500	1000

Table 7: Statistics of the datasets.

an additional regularizer that ensures the distribution of the prompts to be closely correlated with the input x_n. To conclude, at a high level, our method is an alternative to prompt tuning, where we allow the model to treat prompts as distributions rather than deterministic tokens. We rely on a regularization term that forces the model to build more input-predictable distributions for the prompts, and thus provide stability in our (more) stochastic approach.

C Additional Details on Model and Hyperparameters

Training details. The model is pre-trained on a large-collection of Java (210M) and Python (470M) functions and natural language descriptions (47M) from GitHub and StackOverflow. It adopts the BART$_{base}$ Lewis et al. (2020) architecture, which uses 6 layers of Transformer with 768 dimensional hidden states and 12 heads ($\sim$140M parameters). It takes approximately 30 minutes to train a single epoch on a dataset with 100,000 samples on 8 Tesla V100 GPUs.

Number of Clusters. The experimental results show that our approach is not sensitive to the number of clusters. Only at large values (e.g., 18), the performance changes significantly due to model's overfitting. We see a variation of 0.2 in CodeBLEU for k in the range of 6-12, and we choose to set k to 8 for all our experiments.

D Discrete Prefix Analysis

Length of the Discrete Prefix. Number of arguments varies in different datasets. For example, there are an average of 7.3 arguments per sample in the JuIce dataset. Arguments and function names can be mixed, however this requires working with a potentially very long prefix and detracts performance. We experiment with various types of code tokens (Table 8) and their combinations and use argument prefixes based on validation performance.

Study of Different Selections of Prefix Tokens. We provide a study of different selections of discrete prefix tokens on JuICe in Table 8. In addition to using arguments, we also experiment with adapting function names and plot types for prefix tokens. We extract function names by using regular expression, and follow Chen Chen et al. (2021b) to identify plot types by using a predefined mapping (e.g., *"scatter()"* corresponds to scatter plot). The best EM and CodeBLEU scores are achieved by using arguments and plot types, respectively. The result of using function names shows that the inappropriate selection of discrete prefix may lead to a performance drop.

We next experiment with extracting discrete prefixes from input sequences instead of target sequences. This is ordinarily difficult because an input sequence usually does not contain code and is less structured than the target. Fortunately, JuICe dataset contains both programming language in the input and the output, so we can examine the effectiveness of adding informative input code tokens to a target sequence. By doing so, we observe a performance boost in terms of EM and CodeBLUE compared to the PLBART baseline. In this experiment, we also test with forcibly inserting correct prefix tokens into prediction instead of predicting them, but do not observe a significant improvement.

We argue that some tokens extracted from the input can be irrelevant to the target and thus confuse the target prediction.

	EM	CodeBLEU
w/o prefix	4.91	41.61
Function names	4.65	40.29
Plot types	4.71	42.43
Arguments from target	5.17	41.67
Arguments from input	5.07	42.33

Table 8: Results of different selections of discrete prefix on JuICe dataset. "w/o prefix" model can be seen as the baseline.