SMARTPASTE: Learning to Adapt Source Code

Miltiadis Allamanis
Microsoft Research
Cambridge, UK
t-mialla@microsoft.com

Marc Brockschmidt
Microsoft Research
Cambridge, UK
mabrocks@microsoft.com

Abstract

Deep Neural Networks have been shown to succeed at a range of natural language tasks such as machine translation and text summarization. While tasks on source code (*i.e.*, formal languages) have been considered recently, most work in this area does not attempt to capitalize on the unique opportunities offered by its known syntax and structure. In this work, we introduce SMARTPASTE, a first task that requires to use such information. The task is a variant of the program repair problem that requires to adapt a given (pasted) snippet of code to surrounding, existing source code. As first solutions, we design a set of deep neural models that learn to represent the context of each variable location and variable usage in a data flow-sensitive way. Our evaluation suggests that our models can learn to solve the SMARTPASTE task in many cases, achieving 58.6% accuracy, while learning meaningful representation of variable usages.

1 Introduction

The advent of large repositories of source code as well as scalable machine learning methods naturally leads to the idea of "big code", *i.e.*, largely unsupervised methods that support software engineers by generalizing from existing source code. Currently, existing machine learning models of source code capture its shallow, textual structure, *e.g.* as a sequence of tokens [3, 10], as parse trees [7, 11], or as a flat dependency networks of variables [16]. Such models miss out on the opportunity to capitalize on the rich and well-defined semantics of source code. In this work, we take a step to alleviate this by taking advantage of two additional elements of source code: data flow and execution paths. Our key insight is that exposing these semantics explicitly as input to a machine learning model lessens the requirements on amounts of training data, model capacity and training regime and allows us to solve tasks that are beyond the current state of the art.

Some reason 1.只捕获了浅层的结构

key insight：主要解决思路使用明确的语义作为机器学习方法的输入减小对训练数据、模型容量、训练体制的需求

To show how this information can be used, we introduce the SMARTPASTE structured prediction task, in which a larger, existing piece of source code is extended by a new snippet of code and the variables used in the pasted code need to be aligned with the variables used in the context. This task can be seen as a constrained code synthesis task and simultaneously as a useful machine learning-based software engineering tool. To achieve high accuracy on SMARTPASTE, we need to learn representations of program semantics. First an approximation of the semantic role of a variable (*e.g.*, "is it a counter?", "is it a filename?") needs to be learned. Second, an approximation of variable usage semantics (*e.g.*, "a filename is needed here") is required. "Filling the blank element(s)" is related to methods for learning distributed representations of natural language words, such as Word2Vec [12] and GLoVe [14]. However, in our setting, we can learn from a much richer structure, such as data flow information. Thus, SMARTPASTE can be seen as a first step towards learning distributed representations of variable usages in an unsupervised manner. We expect such representations to be valuable in a wide range of tasks, such as code completion ("this is the variable you are looking for"), bug finding ("this is *not* the variable you are looking for"), and summarization ("such variables are usually called `filePath`").

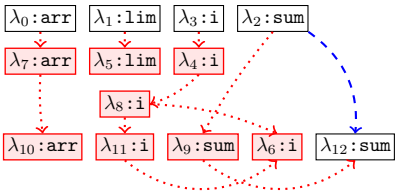
目标：1.近似的语义角色
2.变量的语义用法

与code2vec的相似点，不是从代码文本中学习，而是从更丰富的语义结构中

首先是使用一种无监督的学习方法学习变量用法的分布式表示

```
int SumPositive(int[] arr λ0, int lim λ1) {
    int sum λ2=0;
    for(int i λ3=0; i λ4<lim λ5; i λ6++)
        if (arr λ7[i λ8]>0) sum λ9+=arr λ10[i λ11];
    return sum λ12;
}
```

(a) Example source code, with pasted snippet shaded in green. Tokens corresponding to variables marked by λ_i. The red boxes are the placeholders whose variable needs to be inferred in the SMARTPASTE task (ground truth variable name shown for convenience).



(b) Dataflow diagram for variables in example, using ground truth placeholder choices. Dotted red edges show dataflow that depends on placeholder allocations. Dashed blue edge is dataflow independent of choices.

Figure 1: The snippet (shaded box, left) was pasted into the existing code. Our task is to assign variables to each placeholder (red boxes). This requires inferring the flow of data between placeholders (right). 为每一个占位符分配变量，这需要推断两个占位符之间的数据流来完成

To summarize, our contributions are: (i) We define the SMARTPASTE task as a challenge for machine learning modeling of source code, that requires to learn (some) semantics of programs (*cf.* section 2). (ii) We present five models for solving the SMARTPASTE task by modeling it as a probability distribution over graph structures which represent code's data flow (*cf.* section 3). (iii) We evaluate our models on a large dataset of 4.8 million lines of real-world source code, showing that our best model achieves accuracy of 58.6% in the SMARTPASTE task while learning useful vector representations of variables and their usages (*cf.* section 4). 方法是将其建模为表示代码数据流的图形结构上的概率建模

2 The SMARTPASTE Task

We consider a task beyond standard source code completion in which we want to insert a snippet of code into an existing program and adapt variable identifiers in the snippet to fit the target program (Figure 1). This is a common scenario in software development [4], when developers copy a piece of code from a website (*e.g.* StackOverflow) or from an existing project into a new context. Furthermore, pasting code is a common source of software bugs [15], with more than 40% of Linux porting bugs caused by the inconsistent renaming of identifiers.

While similar to standard code completion, this task differs in a number of important aspects. First, only variable identifiers need to be filled in, whereas many code completion systems focus on a broader task (*e.g.* predicting every next token in code). Second, several identifiers need to be filled in at the same time and thus all choices need to be made synchronously, reflecting interdependencies. This amounts to the structured prediction problem of inferring a graph structure (*cf.* Figure 1b).

首先只需要填写变量的标识符，其次需要填写多个标识符

即所有选择需要同步进行。以反映相互相关性。

这相当于推断图结构的结构化预测问题

Task Description. We view a source code file as a sequence of tokens $t_0 \dots t_N = \mathcal{T}$. The source code contains a set of variables $v_0, v_1 \dots \in \mathbb{V} \subseteq \mathcal{T}$. To simplify the presentation, we assume that the source snippet to be pasted has already been inserted at the target location, and all identifiers in it have been replaced by a set $\mathcal{P}$ of fresh placeholder identifiers (see Figure 1 for an example).

为了简化问题，我们假设粘贴的代码片段已插入目标位置

Thus, our input is a sequence of tokens $t_0 \dots t_N$ with $\{t_{\lambda_1}, \dots, t_{\lambda_K}\} = \mathcal{P}$, and our aim is to find the "correct" assignment $\alpha : \mathcal{P} \to \mathbb{V}$ of variables to placeholders. For training and evaluation purposes, a correct solution is one that simply matches the ground truth, but note that in practice, several possible assignments could be considered correct. 评估标准

Input 是新的占位符p 目标是找到变量到占位符正确的分配

3 Models

In the following, we discuss a sequence of models designed for the SMARTPASTE task, integrating more and more known semantics of the underlying programming language. All models share the concepts of a *context representation* $\mathbf{c}(t)$ of a token t and a *usage representation* $\mathbf{u}(t, v)$ of the usage of a variable v at token t. The models differ in the definitions $\mathbf{c}(t)$ and $\mathbf{u}(t, v)$, but all finally try to maximize the inner product of $\mathbf{c}(t)$ and $\mathbf{u}(t, v)$ for the correct variable assignment v at t.

Notation We use $\mathbb{V}_t \subset \mathbb{V}$ to refer to the set of all variables in scope at the location of t, *i.e.*, those variables that can be legally used at t. Furthermore, we use $\mathcal{U}_{\mathsf{p}}(t, v) \in \mathcal{T} \cup \{\perp\}$ to denote the last occurrence of variable v before t in $\mathcal{T}$ (resp. $\mathcal{U}_{\mathsf{n}}(t, v)$ for the next occurrence), where $\perp$ is used when no previous (resp. next) such token exists. To denote all uses of a variable $v \in \mathbb{V}$, we use $\mathcal{U}(v) \subset \mathcal{T}$. To capture the flow of data through a program, we furthermore introduce the notation $\mathcal{D}_{\mathsf{p}}(t, v) \subseteq \mathcal{T}$, which denotes the set of tokens at which v was possibly last used in an execution of the program (*i.e.* either read from or written to). Similarly, $\mathcal{D}_{\mathsf{n}}(t, v)$ denotes the tokens at which v is next used. Note that $\mathcal{D}_{\mathsf{p}}(t, v)$ (resp. $\mathcal{D}_{\mathsf{n}}(t, v)$) is a set of tokens and extends the notion of $\mathcal{U}_{\mathsf{p}}(t, v)$ (resp. $\mathcal{U}_{\mathsf{n}}(t, v)$) which refers to a single token. Furthermore $\mathcal{D}_{\mathsf{p}}(t, v)$ may include tokens appearing *after* t (resp. $\mathcal{D}_{\mathsf{n}}(t, v)$ may include tokens appearing *before* t) in the case of loops, as it happens for variable `i` in λ_{11} and λ_6 in Figure 1. $\mathcal{D}_{\mathsf{p}}(t, v)$ and $\mathcal{D}_{\mathsf{n}}(t, v)$ for the snippet in Figure 1 are depicted in Figure 2.

Leveraging Variable Type Information We assume a statically typed language and that the source code can be compiled, and thus each variable has a (known) type $\tau(v)$. To use it, we define a learnable embedding function $\mathbf{r}(\tau)$ for known types and additionally define an "UNKTYPE" for all unknown/unrepresented types. We also leverage the rich type hierarchy that is available in many object-oriented languages. For this, we map a variable's type $\tau(v)$ to the set of its supertypes, *i.e.* $\tau^*(v) = \{\tau : \tau(v) \text{ implements type } \tau\} \cup \{\tau(v)\}$. We then compute the type representation $\mathbf{r}^*(v)$ of a variable v as the element-wise maximum of $\{\mathbf{r}(\tau) : \tau \in \tau^*(v)\}$. We chose the maximum here, as it is a natural pooling operation for representing partial ordering relations (such as type lattices). Using all types in $\tau^*(v)$ allows us to generalize to unseen types that implement common supertypes or interfaces. For example, `List<K>` has multiple concrete types (*e.g.* `List<int>`, `List<string>`). Nevertheless, these types implement a common interface (`IList`) and share common characteristics. During training, we randomly select a non-empty subset of $\tau^*(v)$ which ensures training of all known types in the lattice. This acts both like a dropout mechanism and allows us to learn a good representation for types that only have a single known subtype in the training data.

Context Representations To fill in placeholders, we need to be able to learn how they are used. Intuitively, usage is defined by the source code surrounding the placeholder, as it describes what operations are performed on it. Consequently, we define the notion of a *context* of a token t_k as the sequences of C tokens before and after t_k (we use $C = 3$). We use a learnable function f that embeds each token t separately into a vector $f(t)$ and finally compute the *context representation* $\boldsymbol{c}(t)$ using two learnable functions g^p and g^n to combine the token representations as follows.

直观地，用法由占位符周围的源代码定义，因为它描述了对占位符执行的操作。

$$\boldsymbol{c}(t_k) = \mathbf{W}_{\boldsymbol{c}} \cdot \left[g^p\left(f(t_{k-C}), \ldots, f(t_{k-1})\right), g^n\left(f(t_{k+1}), \ldots, f(t_{k+C})\right)\right]$$

Here, $\mathbf{W}_{\boldsymbol{c}}$ is a simple (unbiased) linear layer. Note that we process the representation of preceding and succeeding tokens separately, as the semantics of tokens strongly depends on their position relative to t. In this work, we experiment with a log-bilinear model [13] and a GRU [8] for g. Our embedding function $f(t)$ integrates type information as follows. If t is a variable (*i.e.* $t \in v$) it assigns $\mathbf{r}^*(v)$ to $f(t)$. For each non-variable tokens t, it returns a learned embedding $\mathbf{r}_t$.

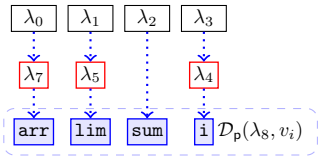
Usage Representations We learn a vector representation $\mathbf{u}(t, v)$ as an approximation of the semantics of a variable v at position t by considering how it has been used *before and after* t. Here, we consider two possible choices of representing usages, namely the *lexical usage representation* and the *data flow usage representation* of a variable.

First, we view source code as a simple sequence of tokens. We define the lexical usage representation $\mathbf{u}^{\mathcal{L}}(t, v)$ of a variable v at placeholder t using up to L (fixed to 14 during training[1]) usages of v around t in lexical order. For this, we use our learnable context representation $\boldsymbol{c}$, and define a sequence of preceding (resp. succeeding) usages of a variable recursively as follows.

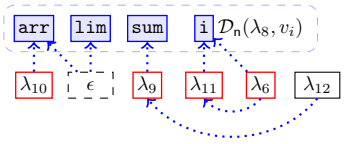
$$\mathcal{U}^{\mathcal{L}}_{\mathsf{p}}(L, t, v) = \begin{cases} \mathcal{U}^{\mathcal{L}}_{\mathsf{p}}(L-1, t', v) \circ \boldsymbol{c}(t') & \text{if } L > 0 \wedge t' = \mathcal{U}_{\mathsf{p}}(t, v) \neq \perp \\ \epsilon & \text{otherwise} \end{cases}$$

$$\mathcal{U}^{\mathcal{L}}_{\mathsf{n}}(L, t, v) = \begin{cases} \boldsymbol{c}(t') \circ \mathcal{U}^{\mathcal{L}}_{\mathsf{n}}(L-1, t', v) & \text{if } L > 0 \wedge t' = \mathcal{U}_{\mathsf{n}}(t, v) \neq \perp \\ \epsilon & \text{otherwise} \end{cases}$$

[1]We set $L = 14$ to capture the 98th percentile of our training data and also allow efficient batching with padding instead of choosing the maximum L in the data.



(a) $\mathcal{D}_\mathsf{p}(\lambda_8, v_i)$ for in-scope variables at λ_8



(b) $\mathcal{D}_\mathsf{n}(\lambda_8, v_i)$ for in-scope variables at λ_8

Figure 2: $\mathcal{D}_\mathsf{p}(\lambda_8, v)$ and $\mathcal{D}_\mathsf{n}(\lambda_8, v)$ for the code in Figure 1. For each in-scope candidate $v \in \mathbb{V}_{\lambda_8}$, a representation is computed using the usage context of that variable before and after that placeholder. Then, the variable $v^* = \arg\max_v (\boldsymbol{c}(\lambda_8))^T \cdot \mathbf{u}(\lambda_8, v)$ is selected for the placeholder. Arrows show the dataflow dependencies of each variable at λ_8 if that variable was to be used at this placeholder.

Here, $\circ$ is sequence composition and ϵ is the empty sequence. Then, we can define $\mathbf{u}^\mathcal{L}(t, v) = h(\mathcal{U}^\mathcal{L}_\mathsf{p}(L, t, v), \mathcal{U}^\mathcal{L}_\mathsf{n}(L, t, v))$, *i.e.* the combination of the representations of the surrounding contexts. We will discuss two choices of h below, namely averaging and a RNN-based model. Note that $\boldsymbol{c}(t)$ is *not* included in either $\mathcal{U}^\mathcal{L}_\mathsf{p}(L, t, v)$ or $\mathcal{U}^\mathcal{L}_\mathsf{n}(L, t, v)$

Our second method for computing $\mathbf{u}(t, v)$ takes the flow of data into account. Instead of using lexically preceding (resp. succeeding) contexts, we consider the data flow relation to identify relevant contexts. Unlike before, there may be several predecessors (resp. successors) of a variable use in the data flow relationship, *e.g.* to reflect a conditional operation on a variable. Thus, we define a *tree* of D preceding contexts $\mathcal{U}^\mathcal{D}_\mathsf{p}(D, t, v)$, re-using our context representation $\boldsymbol{c}$, as a limited unrolling[2] of the data flow graph as follows.

$$\mathcal{U}^\mathcal{D}_\mathsf{p}(D, t, v) = \begin{cases} \{(\boldsymbol{c}(t'_0), \mathcal{U}^\mathcal{D}_\mathsf{p}(D-1, t'_0, v)), \ldots, (\boldsymbol{c}(t'_d), \mathcal{U}^\mathcal{D}_\mathsf{p}(D-1, t'_d, v))\} & \text{if } D > 0 \wedge \mathcal{D}_\mathsf{p}(t, v) = \{t'_0, \ldots, t'_d\} \\ \emptyset & \text{otherwise} \end{cases}$$

The tree of D succeeding contexts $\mathcal{U}^\mathcal{D}_\mathsf{n}(D, t, v)$ is defined analogously. For example, Figure 2 shows $\mathcal{U}^\mathcal{D}_\mathsf{p}(2, \lambda_8, \cdot)$ and $\mathcal{U}^\mathcal{D}_\mathsf{n}(2, \lambda_8, \cdot)$ for all variables in scope at λ_8 of Figure 1. We then compute a representation for the trees using a recursive neural network, whose results are then combined with an unbiased linear layer to obtain $\mathbf{u}^\mathcal{D}(t, v)$. Again, $\boldsymbol{c}(t)$ is *not* in either $\mathcal{U}^\mathcal{D}_\mathsf{p}(D, t, v)$ or $\mathcal{U}^\mathcal{D}_\mathsf{n}(D, t, v)$.

Note that in this way of computing the context, lexically distant variables uses (*e.g.* before a long conditional block or a loop) can be taken into account when computing a representation.

3.1 Learning to Paste

Using the context representation $\boldsymbol{c}(t)$ of the placeholder t and the usage representations $\mathbf{u}(t, v)$ of all variables $v \in \mathbb{V}$ we can now formulate the probability of a single placeholder t being filled by a variable v as the inner product of the two vectors:

$$p(\mathcal{T}[\text{replace } t \text{ by } v]) \propto (\boldsymbol{c}(t))^T \cdot \mathbf{u}(t, v)$$

When considering more than one placeholder, we aim to find an assignment $\alpha : \mathcal{P} \to \mathbb{V}$ such that it maximizes the probability of the code obtained by replacing all placeholders $t \in \mathcal{P}$ according to α at the same time, *i.e.*,

$$\arg\max_\alpha p(\mathcal{T}[\text{replace all } t \in \mathcal{P} \text{ by } \alpha(t)]). \tag{1}$$

As in all structured prediction models, training the model directly on Equation 1 is computationally intractable because the normalization constant requires to compute exponentially (up to $|\mathbb{V}|^{|\mathcal{P}|}$) many different assignments. Thus, during training, we choose to train on a single usage, *i.e.* $\max_{\boldsymbol{\theta}} p(\mathcal{T}[\text{replace } t \text{ by } \alpha(t) \text{ and all others are fixed to ground truth}])$ where $\boldsymbol{\theta}$ are all the parameters of the trained model. However, this objective is still computationally expensive since it requires to compute $\mathbf{u}(t, v)$ for all v of the variably-sized $|\mathbb{V}|$ per placeholder. To circumvent this problem and allow efficient batching, we approximate the normalization constant by using all variables in the current minibatch and train using maximum likelihood.

[2] $D = 15$ during training to covers the 98th percentile in our training data and allows us to batch.

At test time, we need to fill in several placeholders in a given snippet of inserted code. To solve this structured prediction problem, we resort to iterative conditional modes (ICM), where starting from a random allocation α, iteratively for each placeholder t, we pick the variable $v^* = \arg\max_{v \in \mathbb{V}_t} p(\mathcal{T}[\text{replace } t \text{ by } v])$ until the assignment map α converges or we reach a maximum number of iterations. To recover from local optima, we restart the search a few times; selecting the allocation with the highest probability. Note that $\mathcal{U}_{\mathsf{p}}(t, v)$, $\mathcal{U}_{\mathsf{n}}(t, v)$, $\mathcal{D}_{\mathsf{p}}(t, v)$, $\mathcal{D}_{\mathsf{n}}(t, v)$ and thus $\mathbf{u}(t, v)$ change during ICM, as the underlying source code is updated to reflect the last chosen assignment α.

Model Zoo We evaluate 5 different models in this work, based on different choices for the implementation of $\mathbf{u}(t, v)$ and $\boldsymbol{c}(t)$.

- LOC is a baseline using only local type information, *i.e.* $\mathbf{u}(t, v) = \mathbf{r}^*(v)$.
- AVG$\mathcal{G}$ averages over the (variable length) context representations of the lexical context, *i.e.*

$$\mathbf{u}(t, v) = \mathbf{r}^*(v) + \frac{1}{\left|\mathcal{U}^{\mathcal{L}}_{\mathsf{p}}(L, t, v)\right| + \left|\mathcal{U}^{\mathcal{L}}_{\mathsf{n}}(L, t, v)\right|} \left(\sum_i (\mathcal{U}^{\mathcal{L}}_{\mathsf{p}}(L, t, v))_i + \sum_i (\mathcal{U}^{\mathcal{L}}_{\mathsf{n}}(L, t, v))_i \right).$$

- GRU$\mathcal{G}$ uses a combination of the outputs of two GRUs to process the representations of the lexical context, *i.e.*

$$\mathbf{u}(t, v) = \mathbf{W}_{gru} \cdot \left[\text{RNN}^p_{\text{GRU}}(\mathcal{U}^{\mathcal{L}}_{\mathsf{p}}(L, t, v)), \text{RNN}^n_{\text{GRU}}(\mathcal{U}^{\mathcal{L}}_{\mathsf{n}}(L, t, v)) \right],$$

  where $\mathbf{W}_{gru}$ is a learned (unbiased) linear layer. Note that the two RNNs have different learned parameters. The initial state of the RNN_{GRU} is set to $\mathbf{r}^*(v)$.
- GRU$\mathcal{D}$ uses two TreeGRU models (akin to TreeLSTM of Tai et al. [19], but using a GRU cell) over the tree structures $\mathcal{U}^{\mathcal{D}}_{\mathsf{p}}(D, t, v)$ and $\mathcal{U}^{\mathcal{D}}_{\mathsf{n}}(D, t, v)$, where we pool the representations computed for child nodes using an element-wise maximum operation $\text{el}\max$. The state of leafs of the data flow tree are again initialized with the type embedding of v, and thus, we have

$$\mathbf{q}_{\mathsf{p}}(D, t, v) = \begin{cases} \text{el}\max_{t' \in \mathcal{D}_{\mathsf{p}}(t,v)}(\text{GRU}(\boldsymbol{c}(t'), \mathbf{q}_{\mathsf{p}}(D-1, t', v))) & \text{if } D > 0 \wedge \mathcal{D}_{\mathsf{p}}(t, v) \neq \emptyset \\ \mathbf{r}^*(v) & \text{otherwise.} \end{cases}$$

  Analogously, we define $\mathbf{q}_{\mathsf{n}}(\text{D}, t, v)$ and combine them to obtain

$$\mathbf{u}(t, v) = \mathbf{W}_{\mathcal{D}} \cdot \left[\mathbf{q}_{\mathsf{p}}(D, t, v), \mathbf{q}_{\mathsf{n}}(D, t, v) \right],$$

  where $\mathbf{W}_{\mathcal{D}}$ is a learned (unbiased) linear layer.
- H$\mathcal{D}$ is a hybrid between AVG$\mathcal{G}$ and GRU$\mathcal{D}$, which uses another linear layer to combine their usage representations into a single representation of the correct dimensionality.

4 Evaluation

Dataset We collected a dataset for the SMARTPASTE task from open source C# projects on GitHub. To select projects, we picked the top-starred (non-fork) projects in GitHub. We then filtered out projects that we could not (easily) compile in full using Roslyn[3], as we require a compilation to extract precise type information for the code (including those types present in external libraries). Our final dataset contains 27 projects from a diverse set of domains (compilers, databases, ...) with about 4.8 million non-empty lines of code. A full table is shown in Appendix D.

We then created SMARTPASTE examples by selecting snippets of up to 80 syntax tokens (in practice, this means snippets are about 10 statements long) from the source files of a project that are either children of a single AST node (*e.g.* a block or a `for` loop) or are a contiguous sequence of statements. We then replace all variables in the pasted snippet by placeholders. The task is then to infer the variables that were replaced by placeholders.

From our dataset, we selected two projects as our validation set. From the rest of the projects, we selected five projects for UNSEENPROJTEST to allow testing on projects with completely unknown structure and types. We split the remaining 20 projects into train/validation/test sets in the proportion 60-5-35, splitting along files (*i.e.*, *all* examples from one source file are in the same set). We call the test set obtained like this SEENPROJTEST.

[3] `http://roslyn.io`

Table 1: Evaluation of models. UNSEENPROJTEST refers to projects were not part of the train-test split. SEENPROJTEST refers to the test set containing projects that have files in the training set.

	SEENPROJTEST					UNSEENPROJTEST				
	LOC	AVG$\mathcal{G}$	GRU$\mathcal{G}$	GRU$\mathcal{D}$	H$\mathcal{D}$	LOC	AVG$\mathcal{G}$	GRU$\mathcal{G}$	GRU$\mathcal{D}$	H$\mathcal{D}$
Per Placeholder										
Accuracy (%)	41.0	57.8	58.8	56.2	**59.5**	27.9	55.2	52.8	51.3	**56.2**
MRR	0.562	0.719	0.723	0.701	**0.727**	0.476	0.709	0.683	0.664	**0.710**
Type Match (%)	58.0	68.4	70.0	67.8	**70.2**	47.5	64.6	62.8	61.7	**65.4**
Full Snippet Pasting										
Accuracy (%)	47.4	57.9	57.5	55.3	**58.6**	31.7	**54.9**	49.9	49.3	54.5
MRR	0.617	0.711	0.712	0.693	**0.716**	0.491	**0.709**	0.666	0.655	0.700
Ex Match (%)	20.5	30.3	**31.3**	28.8	30.7	8.2	22.5	22.3	21.6	**25.0**
Type Match (%)	54.1	67.3	66.7	64.5	**68.0**	45.4	**62.7**	56.8	56.4	61.2
Type Ex Match (%)	32.0	37.9	**39.3**	36.1	38.7	18.5	**31.1**	27.1	26.5	30.5
Single Placeholder Same-Type Decisions										
PR AUC	0.543	0.819	**0.835**	0.806	0.830	0.494	**0.849**	0.839	0.833	0.833
Precision@10%	87.0	96.5	**99.1**	98.7	97.5	64.0	**99.5**	98.8	98.6	99.0

4.1 Quantitative Evaluation

表1中我们报告了只考虑一个占位符时候的度量标准

MMR最大边缘相关法对文档进行重排序

As a structured prediction problem, there are multiple measures of performance on the task. In the first part of Table 1, we report metrics when considering one placeholder at a time, *i.e.* as if we are pasting a single variable identifier. Accuracy reports the percent of correct predictions, MRR reports the mean reciprocal rank of each prediction. We also measure type correctness, *i.e.* the percent of single-placeholder suggestions that yielded a suggestion of the correct type. In a similar fashion, we present the results when pasting a full snippet. Now, we perform structured prediction over all the placeholders within each snippet, so we can now further compute exact match metrics over all the placeholders. All the models reported here are using a log-bilinear model for computing the context representation $\boldsymbol{c}(t_k)$. Using a GRU for computing $\boldsymbol{c}(t_k)$ yielded slightly worse results for all models. We believe that this is due to optimization issues caused by the increased depth of the network.

Our results in Table 1 show that LOC — as expected — performs worse than all other models, indicating that our other models learn valuable information from the provided usage contexts. Somewhat surprisingly, our relatively simple AVG$\mathcal{G}$ already performs well. On the other hand, GRU$\mathcal{D}$ performs worse than models not taking the flow of data into account. We investigated this behavior more closely and found that the lexical context models can often profit from observing the use of variables in other branches of a conditional statement (*i.e.*, peek at the `then` case when handling a snippet in the `else` branch). Consequently, H$\mathcal{D}$, which combines data flow information with all usages always achieves high performance using both kinds of information. Finally, to evaluate the need for type information, we run the experiments removing all type information. This — on average — resulted to an 8% reduced performance on the SMARTPASTE task on all models.

Same-Type Decisions So far, we considered the SMARTPASTE task where for each placeholder the neural networks consider all variables in scope. However, if we assume that we know the desired type of the placeholder, we can limit the set of suggestions. The last set of metrics in Table 1 evaluate this scenario, *i.e.* the suggestion performance within placeholders that have two or more type-correct possible suggestions. In our dataset, there are on average 5.4 (median 2) same-type variables in-scope per placeholder used in this evaluation. All networks (except LOC) achieve high precision-recall with a high AUC. This implies that our networks do *not* just learn typing information. Furthermore, for 10% recall our best model achieves a precision of 99.1%. First, this suggests that these models can be used as a high-precision method for detecting bugs caused by copy-pasting or porting that the code's existing type system would fail to catch. Additionally, this indicates that our model have learned a probabilistic refinement of the existing type system, *i.e.*, that they can distinguish counters from other `int` variables; file names from other `strings`; *etc.*

Generalization to new projects Generalizing across a diverse set of source code projects with different domains is an important challenge in machine learning. We repeat the evaluation using the UNSEENPROJTEST set stemming from projects that have no files in the training set. The right side of Table 1 shows that our models still achieve good performance, although it is slightly lower compared to SEENPROJTEST, especially when matching variable types. This is expected since the type lattice is mostly unknown in UNSEENPROJTEST. We believe that some of the most important

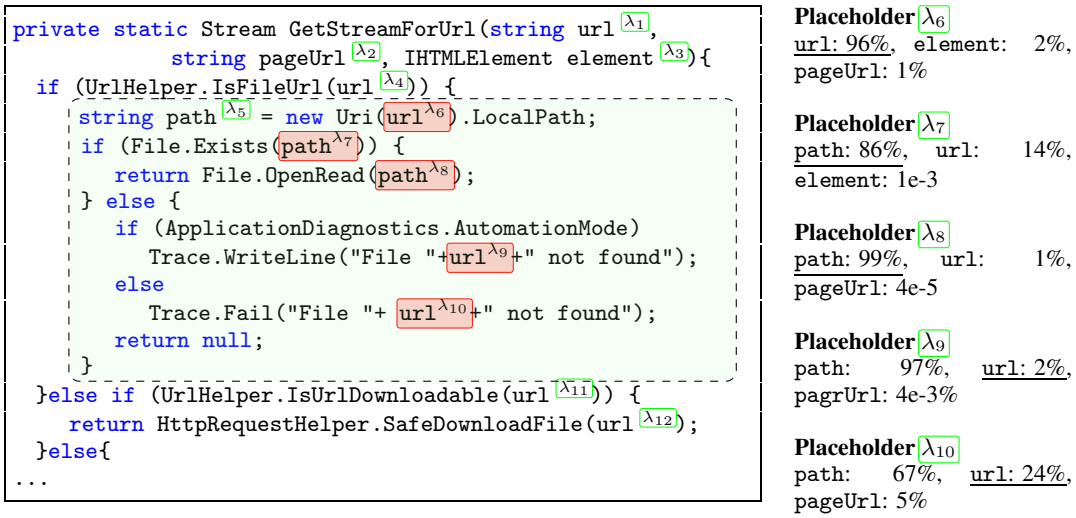


Figure 3: SMARTPASTE suggestion on snippet of the SEENPROJTEST set. H$\mathcal{D}$ suggests all the red placeholders (λ_6 to λ_{10}) in the shaded area (λ_5 is a declaration). The probability for each placeholder is shown on the right. Note that there are multiple `string` variables in scope (`url`, `pageUrl`, `path`). However, H$\mathcal{D}$ learns usage patterns (*e.g.* `url` is a parameter of `IsFileUrl`) to assign different representations to each variable usage. This allows us to discriminate between `path`, `url` and `pageUrl`. The model ranks second the ground truth for λ_9, λ_{10} suggesting `path` instead, which nevertheless seems reasonable. Variable names are *not* used in the model, but are shown for convenience. Additional visualizations are available in Appendix C.

issues when transferring to new domains is the fact that projects have significantly different type hierarchies and that the vocabulary used (*e.g.* by method names) is very different from the training projects.

4.2 Qualitative Evaluation

We show an example of the SMARTPASTE task in Figure 3, where we can observe that the model learns to discriminate both among variables with different types (`elements` of type `IHTMLElement` is not confused with `string` variables) as well as assigning more fine-grained semantics (`url` and `path` are treated separately) as implied by the results for our same-type scenario above.

In Figure 4, we show placeholders that have highly similar usage context representations $\mathbf{u}(t, v)$. Qualitatively, Figure 4 and the visualizations in Appendix B suggest that the learned representations can be used as a learned similarity metric for variable usage semantics. These representations learn protocols and conventions such as "after accessing `X`, we should access `Y`" or the need to conditionally check a property, as shown in Figure 4.

We observed a range of common problems. Most notably, variables that are declared but not explicitly initialized (*e.g.* as a method parameter) cause the usage representation to be uninformative, grouping all such declarations into the same representation. The root cause is the limited information available in the context representations. Local optima in ICM and UNK tokens also are common.

5 Related Work

Our work builds upon the recent field of using machine learning for source code artifacts. Recent research has lead to language models of code that try to model the whole code. Bhoopchand et al. [6], Hindle et al. [10] model the code as a sequence of tokens, while Maddison and Tarlow [11], Raychev et al. [17] model the syntax tree structure of code. All the work on language models of code find that predicting variable and method identifiers is one of biggest challenges in the task. We are not aware of any models that attempt to use data flow information for variables.

```
...
_generatedCodeAnalysisFlagsOpt = generatedCodeAnalysisFlagsOpt;
...
context.RegisterCompilationStartAction(this.OnCompilationStart);
if (  ?  .HasValue)
    context.ConfigureGeneratedCodeAnalysis( _generatedCodeAnalysisFlagsOpt .Value);
...
```

```
...
var  symbolAndProjectId  = await definition.TryRehydrateAsync(
    _solution, _cancellationToken).ConfigureAwait(false);
if (!  ?  .HasValue) return;
lock (_gate){
    _definitionMap[definition]= symbolAndProjectId .Value;
}
...
```

Figure 4: Placeholders (in black ?) with similar usage embeddings $\mathbf{u}(t, v)$. Both `_generatedCodeAnalysisFlagsOpt` and `symbolAndProjectId` implement the `Nullable` interface. Note that the local context `if(  ?  .HasValue)` is *not* used when computing $\mathbf{u}(t, v)$ but data flow information of the other usages is used (marked in yellow). In this example, the model learns a common representation for `Nullables` that are assigned and then conditionally used by accessing the `.HasValue` property. The formatting of the snippets has been changed for space saving. More examples can be found in Appendix A. 模型通过访问.hasValue属性来学习分配给的NUllables 的通用表示形式，然后有条件的使用它们

Closest to our work is the work of Allamanis et al. [2] who learn distributed representations of variables using all their usages to predict their names. However, they do not use data flow information and only consider semantically equivalent renaming of variables (α-renaming). Finally, the work of Raychev et al. [16] is also relevant, as it uses a dependency network between variables. However, all variable usages are deterministically known beforehand (as the code is complete and remains unmodified), as in Allamanis et al. [1, 2].

Our work is remotely related to work on program synthesis using sketches [18] and automated code transplantation [5]. However, these approaches require a set of specifications (*e.g.* input-output examples, test suites) to complete the gaps, rather than statistics learned from big code. These approaches can be thought as complementary to ours, since we learn to statistically complete the gaps without any need for specifications, by learning common dataflow structure from code.

Our problem has also similarities with coreference resolution in NLP and methods for the structured prediction of graphs and — more commonly — trees. However, given the different characteristics of the problems, such as the existence of exact execution path information, we are not aware of any work that would be directly relevant. Somewhat similar to our work, is the work of Clark and Manning [9], who create a neural model that learns to rank pairs of clusters of mentions to either merge them into a single co-reference entity or keep them apart.

6 Discussion & Conclusions

方向是要利用好编程语言已经定义好的丰富语义，和有效的分析工具提取这些信息

Although source code is well understood and studied within other disciplines such as programming language research, it is a relatively new domain for deep learning. It presents novel opportunities compared to textual or perceptual data, as its (local) semantics are well-defined and rich additional information can be extracted using well-known, efficient program analyses. On the other hand, integrating this wealth of structured information poses an interesting challenge. Our SMARTPASTE task exposes these opportunities, going beyond more simple tasks such as code completion. We consider it as a first proxy for the core challenge of learning the *meaning* of source code, as it requires to probabilistically refine standard information included in type systems.

不足，没有考虑到标识的名字

We see a wealth of opportunities in the research area. To improve on our performance on the SMARTPASTE task, we want to extend our models to additionally take identifier names into account, which

are obviously rich in information. Similarly, we are interested in exploring more advanced tasks such as bug finding, automatic code reviewing, *etc.*

Acknowledgments

We would like to thank Alex Gaunt for his valuable comments and suggestions.

References

[1] M. Allamanis, E. T. Barr, C. Bird, and C. Sutton. Learning natural coding conventions. In *International Symposium on Foundations of Software Engineering (FSE)*, 2014.

[2] M. Allamanis, E. T. Barr, C. Bird, and C. Sutton. Suggesting accurate method and class names. In *Foundations of Software Engineering (FSE)*, 2015.

[3] M. Allamanis, H. Peng, and C. Sutton. A convolutional attention network for extreme summarization of source code. In *Proceedings of The 33rd International Conference on Machine Learning*, pages 2091–2100, 2016.

[4] S. Amann, S. Proksch, S. Nadi, and M. Mezini. A study of Visual Studio usage in practice. In *International Conference on Software Analysis, Evolution, and Reengineering (SANER)*, 2016.

[5] E. T. Barr, M. Harman, Y. Jia, A. Marginean, and J. Petke. Automated software transplantation. In *Proceedings of the 2015 International Symposium on Software Testing and Analysis*, 2015.

[6] A. Bhoopchand, T. Rocktäschel, E. Barr, and S. Riedel. Learning Python code suggestion with a sparse pointer network. *arXiv preprint arXiv:1611.08307*, 2016.

[7] P. Bielik, V. Raychev, and M. Vechev. PHOG: probabilistic model for code. In *International Conference on Machine Learning*, 2016.

[8] K. Cho, B. van Merriënboer, D. Bahdanau, and Y. Bengio. On the properties of neural machine translation: Encoder–decoder approaches. *Syntax, Semantics and Structure in Statistical Translation*, 2014.

[9] K. Clark and C. D. Manning. Improving coreference resolution by learning entity-level distributed representations. *arXiv preprint arXiv:1606.01323*, 2016.

[10] A. Hindle, E. T. Barr, Z. Su, M. Gabel, and P. Devanbu. On the naturalness of software. In *International Conference on Software Engineering (ICSE)*, 2012.

[11] C. J. Maddison and D. Tarlow. Structured generative models of natural source code. In *International Conference on Machine Learning (ICML)*, 2014.

[12] T. Mikolov, I. Sutskever, K. Chen, G. S. Corrado, and J. Dean. Distributed representations of words and phrases and their compositionality. In *Advances in neural information processing systems*, 2013.

[13] A. Mnih and Y. W. Teh. A fast and simple algorithm for training neural probabilistic language models. In *Proceedings of the 29th International Conference on Machine Learning*, 2012.

[14] J. Pennington, R. Socher, and C. D. Manning. GloVe: Global vectors for word representation. In *EMNLP*, 2014.

[15] B. Ray, M. Kim, S. Person, and N. Rungta. Detecting and characterizing semantic inconsistencies in ported code. In *International Conference on Automated Software Engineering (ASE)*, 2013.

[16] V. Raychev, M. Vechev, and A. Krause. Predicting program properties from Big Code. In *ACM SIGPLAN Notices*, 2015.

[17] V. Raychev, P. Bielik, and M. Vechev. Probabilistic model for code with decision trees. In *International Conference on Object-Oriented Programming, Systems, Languages, and Applications*, 2016.

[18] A. Solar-Lezama. *Program synthesis by sketching*. University of California, Berkeley, 2008.

[19] K. S. Tai, R. Socher, and C. D. Manning. Improved semantic representations from tree-structured long short-term memory networks. *arXiv preprint arXiv:1503.00075*, 2015.

A Per Placeholder Suggestion Samples

Below we list a set of sample same-type decisions made when considering one placeholder at a time. Some code comments and formatting have been altered for typesetting reasons. The ground truth choice is underlined.

Sample 1

```
private static DataTable CreateDataTable(int cols, string colNamePrefix)
{
    var table = new DataTable();
    for (int i = 1; i <= cols; i++)
    {
        table.Columns.Add(new DataColumn() { ColumnName = colNamePrefix + #1 ,
                                             DefaultValue = #2 });
    }
    table.Rows.Add(table.NewRow());
    return table;
}
```

#1 i: 84%, cols: 16%
#2 i: 53%, cols: 47%

Sample 2

```
public void A_VectorClock_must_not_happen_before_itself()
{
    var clock1 = VectorClock.Create();
    var clock2 = VectorClock.Create();

    ( #1 != #2 ).Should().BeFalse();
}
```

#1 clock1: 44%, clock2: 56%
#2 clock1: 9%, clock2: 91%

Sample 3

```
public MergeHub(int perProducerBufferSize)
{
    if ( #1 <= 0)
        throw new ArgumentException("Buffer size must be positive", nameof( #2 ));

    _perProducerBufferSize = perProducerBufferSize;
    DemandThreshold = perProducerBufferSize/2 + perProducerBufferSize%2;
    Shape = new SourceShape<T>(Out);
}
```

#1 perProducerBufferSize: 100%, _perProducerBufferSize: 2e-4, DemandThreshold: 1e-6
#2 perProducerBufferSize: 100%, _perProducerBufferSize: 3e-3, DemandThreshold: 2e-3

Sample 4

```
public Task UpdateRuntimeStatistics(SiloAddress siloAddress,
                                    SiloRuntimeStatistics siloStats)
{
    if (logger.IsVerbose)
        logger.Verbose("UpdateRuntimeStatistics from {0}", siloAddress);
    if (this.siloStatusOracle.GetApproximateSiloStatus(siloAddress)
                                                    != SiloStatus.Active)
        return Task.CompletedTask;

    SiloRuntimeStatistics old;
    // Take only if newer.
    if (periodicStats.TryGetValue(siloAddress, out old)
                    && old.DateTime > siloStats.DateTime)
        return Task.CompletedTask;

    #1 [siloAddress] = #2 ;
    NotifyAllStatisticsChangeEventsSubscribers(siloAddress, #3 );
    return Task.CompletedTask;
}
```

#1 periodicStats: 94%, PeriodicStats: 6%
#2 siloStats: 89%, old: 11%
#3 old: 54%, siloStats: 46%

Sample 5

```
public override BoundNode VisitLocal(BoundLocal node)
{
    LocalSymbol localSymbol = node.LocalSymbol;
    CheckAssigned(localSymbol, node.Syntax);

    if (localSymbol.IsFixed &&
        (this. #1 .MethodKind == MethodKind.AnonymousFunction ||
            this. #2 .MethodKind == MethodKind.LocalFunction) &&
         #3 .Contains(localSymbol))
    {
        Diagnostics.Add(ErrorCode.ERR_FixedLocalInLambda,
                        new SourceLocation(node.Syntax), localSymbol);
    }
    return null;
}
```

#1 currentMethodOrLambda: 100%, topLevelMethod: 4e-3
#2 currentMethodOrLambda: 100%, topLevelMethod: 2e-4
#3 _writtenVariables: 60%,_capturedVariables: 40%

Sample 6

```
private IDbContextServices InitializeServices()
{
    if ( #1 )
    {
        throw new InvalidOperationException(CoreStrings.RecursiveOnConfiguring);
    }
...
```

#1 _initializing: 73%, _disposed: 27%

Sample 7

```
public static IMutableForeignKey GetOrAddForeignKey(
            [NotNull] this IMutableEntityType entityType,
            [NotNull] IReadOnlyList<IMutableProperty> properties,
            [NotNull] IMutableKey principalKey,
            [NotNull] IMutableEntityType principalEntityType)
{
    Check.NotNull( #1 , nameof( #2 ));

    return  #3 .FindForeignKey(properties, principalKey,  #4 )
            ?? #5 .AddForeignKey(properties, principalKey,  #6 );
}
```

#1 entityType: 100%, principalEntityType: 4e-3
#2 entityType: 100%, principalEntityType: 3e-4
#3 entityType: 78%, principalEntityType: 22%
#4 principalEntityType: 100%, entityType: 2e-3
#5 principalEntityType: 60%, entityType: 30%
#6 entityType: 99%, principalEntityType: 1%

Sample 8

```
public string URL
{
    get
    {
        if ( #1 == null)
        {
            // Read the URL into a string
            Stream stream = (Stream)m_dataObject.GetData(DataFormatsEx.URLFormat);
            StreamReader reader = new StreamReader(stream);

            using (reader)
            {
                 #2 = reader.ReadToEnd().Trim((char)0);
            }
        }
        return  #3 ;
    }
}
```

#1 m_url: 90%, m_title: 5%, URL: 4%, Title: 1%
#2 m_url: 84%, Title:13%, m_title: 1%, URL: 1%,
#3 m_url: 99%, m_title: 4e-3, URL: 3e-3, Title:6e-4

Sample 9

```
internal static byte[] UrlEncodeToBytes(byte[] bytes, int offset, int count)
{
    if (bytes == null)
        throw new ArgumentNullException("bytes");

    int blen = bytes.Length;
    if ( #1 == 0)
        return ArrayCache.Empty<byte>();

    if ( #2 < 0 || #3 >= #4 )
        throw new ArgumentOutOfRangeException("offset");
...
```

#1 blen: 85%, offset: 9%, count: 6%
#2 offset: 43%, blen: 36%, count: 21%
#3 offset: 76%, blen: 13%, count: 11%
#4 count: 60%, offset: 31%, blen: 10%

Sample 10

```
private static List<UsingDirectiveSyntax> AddUsingDirectives(
          CompilationUnitSyntax root, IList<UsingDirectiveSyntax> usingDirectives)
{
    // We need to try and not place the using inside of a directive if possible.
    var usings = new List<UsingDirectiveSyntax>();
    var endOfList = root.Usings.Count - 1;
    var startOfLastDirective = -1;
    var endOfLastDirective = -1;
    for (var i = 0; #1 < root.Usings.Count; #2 ++)
    {
        if (root.Usings[ #3 ].GetLeadingTrivia()
                    .Any(trivia => trivia.IsKind(SyntaxKind.IfDirectiveTrivia)))
        {
            #4 = #5 ;
        }

        if (root.Usings[ #6 ].GetLeadingTrivia()
                    .Any(trivia => trivia.IsKind(SyntaxKind.EndIfDirectiveTrivia)))
        {
            #7 = #8 ;
        }
    }
...
```

#1 i: 98%, endOfList: 1%, startOfLastDirective: 2e-3, endOfLastDirective: 3e-3
#2 i: 99%, endOfList: 2e-3, startOfLastDirective: 6e-3, endOfLastDirective: 1e-3
#3 i: 100%, endOfList: 1e-4, startOfLastDirective: 3e-4%, endOfLastDirective: 5e-5
#4 endOfLastDirective: 58%, startOfLastDirective: 30%, endOfList: 1%, i: 3e-3
#5 endOfLastDirective: 77%, startOfLastDirective: 12%, i: 6%, endOfList: 5%
#6 i: 100%, endOfList: 1e-3, startOfLastDirective: 2e-4, endOfLastDirective: 5e-4
#7 endOfLastDirective: 52%, startOfLastDirective: 37%, endOfList: 1%, i: 3e-3,
#8 i: 53%, endOfLastDirective: 27%, startOfLastDirective: 13%, endOfList: 7%

B Nearest Neighbor of Usage Representations

Here we show pairs of nearest neighbors based on the cosine similarity of the learned representations $\mathbf{u}(t, v)$. Each placeholder t is marked as ? and all usages of v are marked in yellow (*i.e.* variableName). Although names of variables are shown for convenience, they are *not* used (only their types — if known — is used). This is a set of hand-picked examples showing good and bad examples. A brief description follows after each pair.

Sample 1

```
public void SetDateTime(string year, string month, string day)
{
    string time = "";
    if (year.Contains(":"))
    {
        ? = year;
        year = DateTime.Now.Year.ToString();
        TimeInfo = true;
    }

    DateTime = DateTime.Parse(string.Format("{0}/{1}/{2} {3}", year,
                                month, day, time));
    DateTime = DateTime.ToLocalTime();
}
```

```
public void MakeMultiDirectory(string dirName)
{
    string path = "";
    string[] dirs = dirName.Split('/');
    foreach (string dir in dirs)
    {
        if (!string.IsNullOrEmpty(dir))
        {
            ? = URLHelpers.CombineURL(path, dir);
            MakeDirectory(URLHelpers.CombineURL(Options.Account.FTPAddress, path));
        }
    }

    WriteOutput("MakeMultiDirectory: " + dirName);
}
```

▷ Usage context where a string has been initialized to blank but may be reassigned before it is used.

Sample 2

```
...
FtpWebRequest request = (FtpWebRequest)WebRequest.Create(url);
    ?  .Proxy = Options.ProxySettings;
request.Method = WebRequestMethods.Ftp.ListDirectory;
request.Credentials = new NetworkCredential(Options.Account.Username,
            Options.Account.Password);
request.KeepAlive = false;
request.Timeout = 10000;
request.UsePassive = !Options.Account.IsActive;

using (WebResponse response = request.GetResponse()) {
...
```

```
...
FtpWebRequest request = (FtpWebRequest)WebRequest.Create(url);
    ?  .Proxy = Options.ProxySettings;
request.Method = WebRequestMethods.Ftp.RemoveDirectory;
request.Credentials = new NetworkCredential(Options.Account.Username,
            Options.Account.Password);
request.KeepAlive = false;

request.GetResponse();
...
```

▷ Similar protocols when using an object.

Sample 3

```
...
var addMethod = @event.AddMethod;
Assert.Equal(voidType,  ?  .ReturnType);
Assert.True(addMethod.ReturnsVoid);
Assert.Equal(1, addMethod.ParameterCount);
Assert.Equal(eventType, addMethod.ParameterTypes.Single());
...
```

```
...
var removeMethod = @event.RemoveMethod;
Assert.Equal(voidType,  ?  .ReturnType);
Assert.True(removeMethod.ReturnsVoid);
Assert.Equal(1, removeMethod.ParameterCount);
Assert.Equal(eventType, removeMethod.ParameterTypes.Single());
...
```

▷ These two placeholders have — by definition — identical representations.

Sample 4

```
...
int index = flpHotkeys.Controls.GetChildIndex(Selected);

int newIndex;

if ( ? == 0)
{
    newIndex = flpHotkeys.Controls.Count - 1;
}
else
{
    newIndex = index - 1;
}

flpHotkeys.Controls.SetChildIndex(Selected, newIndex);
manager.Hotkeys.Move(index, newIndex);
...
```

```
...
if (Selected != null && flpHotkeys.Controls.Count > 1)
{
    int index = flpHotkeys.Controls.GetChildIndex(Selected);

    int newIndex;

    if ( ? == flpHotkeys.Controls.Count - 1)
    {
        newIndex = 0;
    }
    else
    {
        newIndex = index + 1;
    }

    flpHotkeys.Controls.SetChildIndex(Selected, newIndex);
    manager.Hotkeys.Move(index, newIndex);
...
```

Sample 5

```
int index = flpHotkeys.Controls.GetChildIndex(Selected);
int newIndex;
if (index == 0)
{
    ? = flpHotkeys.Controls.Count - 1;
}
else
{
    newIndex = index - 1;
}
flpHotkeys.Controls.SetChildIndex(Selected, newIndex);
manager.Hotkeys.Move(index, newIndex);
```

```
int index = flpHotkeys.Controls.GetChildIndex(Selected);
int newIndex;
if (index == 0)
{
    newIndex = flpHotkeys.Controls.Count - 1;
}
else
{
    ? = index - 1;
}
flpHotkeys.Controls.SetChildIndex(Selected, newIndex);
manager.Hotkeys.Move(index, newIndex);
```

▷ Because of the dataflow, these two placeholders (one in each branch of the `if-else`) have identical representations in the dataflow model, and have very similar representations in other models.

Sample 6

```
_generatedCodeAnalysisFlagsOpt = generatedCodeAnalysisFlagsOpt;
...
context.RegisterCompilationStartAction(this.OnCompilationStart);

if ( ? .HasValue)
{
    // Configure analysis on generated code.
    context.ConfigureGeneratedCodeAnalysis( _generatedCodeAnalysisFlagsOpt .Value);
}
...
```

```
...
var symbolAndProjectId = await definition.TryRehydrateAsync(
    _solution, _cancellationToken).ConfigureAwait(false);

if (! ? .HasValue)
{
    return;
}

lock (_gate)
{
    _definitionMap[definition] = symbolAndProjectId .Value;
}
...
```

▷ Our model learns a similar representation for the placeholder between the locations where a `Nullable` variable is assigned and used, which corresponds to a check on the `.HasValue` property.

Sample 7

```
var analyzers = new DiagnosticAnalyzer[] { new ConcurrentAnalyzer(typeNames) };
var expected = new DiagnosticDescription[typeCount];
for (int i = 0; ? < typeCount; i ++)
{
    var typeName = $"C{ i + 1}";
    expected[ i ] = Diagnostic(ConcurrentAnalyzer.Descriptor.Id, typeName)
        .WithArguments(typeName)
        .WithLocation( i + 2, 7);
}
```

```
var builder = new StringBuilder();
var typeCount = 100;
var typeNames = new string[typeCount];
for (int i = 1; ? <= typeCount; i ++)
{
    var typeName = $"C{ i }";
    typeNames[ i - 1] = typeName;
    builder.Append($"\r\nclass {typeName} {{ }}");
}
```

▷ The model learns — unsurprisingly — a very similar representation of the loop control variable `i` at the location of the bound check. Generalizing over varying loops.

Sample 8

```
...
if (!   ?   )
{
    if (disposeManagedResources)
    {
        _resizerControl.SizerModeChanged +=
                    new SizerModeEventHandler(resizerControl_SizerModeChanged);
        _resizerControl.Resized -= new EventHandler(resizerControl_Resized);
        _dragDropController.Dispose();
    }

     _disposed = true;
}
...
```

```
...
if (!   ?   )
{
    _enableRealTimeWordCount = Settings.GetBoolean(SHOWWORDCOUNT, false);
    _enableRealTimeWordCountInit = true;
}
return _enableRealTimeWordCount;
...
```

▷ Similar representations for booleans that will be assigned to `true` within a branch.

Sample 9

```
...
SmartContentSelection selection = EditorContext.Selection as SmartContentSelection;
if (   ?   != null)
{
    return selection.HTMLElement.sourceIndex == HTMLElement.sourceIndex;
}
else
{
    return false;
}
...
```

```
...
foreach (LiveClipboardFormat format in formats)
{
 ContentSourceInfo contentSource = FindContentSourceForLiveClipboardFormat(format);
 if (   ?   != null)
    return contentSource;
}
...
```

▷ Representation for elements that will be returned but only one one path.

Sample 10

```
...
Rectangle elementRect = ElementRectangle;
_resizerControl.VirtualLocation = new Point(
            elementRect.X - ResizerControl.SIZERS_PADDING,
            ?.Y - ResizerControl.SIZERS_PADDING);
...
```

```
...
Rectangle rect = CalculateElementRectangleRelativeToBody(HTMLElement);
IHTMLElement body = (HTMLElement.document as IHTMLDocument2).body;

_dragBufferControl.VirtualSize = new Size(body.offsetWidth, body.offsetHeight);
_dragBufferControl.VirtualLocation = new Point(-rect.X, -?.Y);
_dragBufferControl.Visible = true;
...
```

▷ Protocol of accesses for `Rectangle` objects. After `X` has been accessed then the variable has the same representation (implied that `Y` is highly likely to be accessed next)

Sample 11

```
...
if (parameters != null)
{
    for (int i = 0; i < parameters.Length; i += 2)
    {
        string name = ?[i];
        string val = parameters[i + 1];
        if (!cullMissingValues || (val != null && val != string.Empty))
            Add(name, val);
    }
}
...
```

```
...
string[] refParams = value.Split(commaSeparator);

if (refParams.Length != 2 || string.IsNullOrEmpty(refParams[0])
                  || string.IsNullOrEmpty(refParams[1]))
    throw new ArgumentException("Reference path is invalid.");

ModuleName = ?[0];
ResourceId = int.Parse(refParams[1]);

referencePath = value;
...
```

▷ Similar representation for first array access after bound checks.

Sample 12

```
...
public static string GetHostName(string url)
{
    if (!IsUrl( ? ))
        return null;
    return new Uri(url).Host;
}
...
```

```
...
public static bool IsUrlLinkable(string url)
{
    if (UrlHelper.IsUrl( ? ))
    {
        Uri uri = new Uri(url);
        foreach (string scheme in NonlinkableSchemes)
            if (uri.Scheme == scheme)
                return false;

    }
    return true;
}
...
```

▷ Similar representations because of learned pattern when parsing URIs.

Sample 13

```
...
private void WriteEntry(string message, string category, string stackTrace)
{
    // Obtain the DateTime the message reached us.
    DateTime dateTime = DateTime.Now;

    // Default the message, as needed.
    if (message == null || message.Length == 0)
        message = "[No Message]";

    // Default the category, as needed.
    if (category == null || category.Length == 0)
        ?        = "None";

    int seqNum = Interlocked.Increment(ref sequenceNumber);

    DebugLogEntry logEntry = new DebugLogEntry(facility, processId, seqNum,
                                               dateTime, message, category, stackTrace);
...
```

```
...
private void WriteEntry(string message, string category, string stackTrace)
{
    // Obtain the DateTime the message reached us.
    DateTime dateTime = DateTime.Now;

    // Default the message, as needed.
    if (message == null || message.Length == 0)
        ?        = "[No Message]";

    // Default the category, as needed.
    if (category == null || category.Length == 0)
        category = "None";

    int seqNum = Interlocked.Increment(ref sequenceNumber);

    DebugLogEntry logEntry = new DebugLogEntry(facility, processId, seqNum,
                                               dateTime, message, category, stackTrace);
...
```

▷ The variables `message` and `category` (in the same snippet of code) have similar representations. This is a source of confusion for our models.

Sample 14

```
...
foreach (string file in files)
{
    string[] chunks = ? .Split(INTERNAL_EXTERNAL_SEPARATOR);
    switch (chunks[0])
    {
...
```

```
...
Uri uri = new Uri(url);
foreach (string scheme in NonlinkableSchemes)
    if (uri.Scheme == ? )
        return false;
...
```

▷ Limited context (*e.g.* only declaration) causes variables to have similar usage representations. In the examples `file` and `scheme` are defined and used only once. This is a common source of confusion for our models.

C Full Snippet Pasting Samples

Below we present some of the suggestions when using the full SMARTPASTE structured prediction. The variables shown at each placeholder correspond to the ground truth. Underlined tokens represent UNK tokens. The top three allocations are shown as well as the ground truth (if it is *not* in the top 3 suggestions). Red placeholders are the placeholders that need to be filled in when pasting. All other placeholders are marked in superscript next to the relevant variable.

Sample 1

```
...
charsLeft^λ1 = 0;
while (p^λ2.IsRightOf(selection^λ3.Start))
{
    charsLeft^λ4++;
    p^λ5.MoveUnit(_MOVEUNIT_ACTION.MOVEUNIT_PREVCHAR);
}
...
```

λ_1 charsLeft: 87%, movesRight: 8%, p: 5%

λ_2 p: 96%, selection: 4%, bounds: 1e-3

λ_3 selection: 89%, bounds: 1%, p: 8e-3

λ_4 movesRight: 66%, charsLeft: 16%, p: 1%

λ_5 p: 83%, selection: 11%, bounds: 6%

Sample 2

```
...
HttpWebResponse response λ0 = null;
XmlDocument xmlDocument λ1 = new XmlDocument();
try
{
    using (Blog blog λ3 = new Blog(_blogId λ4))
        response λ5 = blog λ6.SendAuthenticatedHttpRequest(notificationUrl λ7, 10000);

    // parse the results
    xmlDocument λ8.Load(response λ9.GetResponseStream());
}
catch (Exception)
{
    throw;
}
finally
{
    if (response λ10 != null)
        response λ11.Close();
}
...
```

λ_4 _hostBlogId: 12%, BlogId: 10%, _buttonId: 10%, _blogId: 1%

λ_5 response: 86%, xmlDocument: 5%, notificationUrl: 3%

λ_6 xmlDocument: 84%, blog: 12%, response: 2%

λ_7 NotificationPollingTime: 95%, CONTENT_DISPLAY_SIZE: 2%, notificationUrl: 1%

λ_8 xmlDocument: 100%, response: 9e-4, _buttonDescription: 4e-4

λ_9 response: 65%, xmlDocument: 30%, _hostBlogId: 4%

λ_{10} response: 90%, _blogId: 3%, CurrentImage: 9e-3

λ_{11} response: 98%, _settingKey: 1%, xmlDocument: 9e-3

Sample 3

```
...
protected override void Dispose(bool disposing λ1)
{
    if (disposing λ2)
    {
        if (components λ3 != null)
            components λ4.Dispose();
    }
    base.Dispose(disposing λ5);
}
...
```

λ_2 disposing: 100%, commandIdentifier: 4e-4, components: 1e-4

λ_3 components: 100%, disposing: 3e-5, commandIdentifier: 2e-5

λ_4 components: 100%, disposing: 9e-7, CommandIdentifier: 6e-9

λ_5 disposing: 100%, components: 3e-5, CommandIdentifier: 2e-5

Sample 4

```
...
tmpRange [λ1].Start.MoveAdjacentToElement(startStopParent [λ2],
                                          _ELEMENT_ADJACENCY.ELEM_ADJ_BeforeBegin);
if (tmpRange [λ3].IsEmptyOfContent())
{
    tmpRange [λ4].Start.MoveToPointer(selection [λ5].End);
    IHTMLElement endStopParent [λ6] = tmpRange [λ7].Start.GetParentElement(stopFilter [λ8]);

    if (endStopParent^λ9 != null
        && startStopParent^λ10.sourceIndex == endStopParent^λ11.sourceIndex)
    {
        tmpRange^λ12.Start
                .MoveAdjacentToElement(endStopParent^λ13,
                                       _ELEMENT_ADJACENCY.ELEM_ADJ_BeforeEnd);
        if (tmpRange^λ14.IsEmptyOfContent())
        {
            tmpRange^λ15.MoveToElement(endStopParent^λ16, true);
            if (maximumBounds^λ17.InRange(tmpRange^λ18)
                                    && !(endStopParent^λ19 is IHTMLTableCell))
            {
                deleteParentBlock^λ20 = true;
            }
        }
    }
}
...
```

λ_9 startStopParent: 97%, styleTagId: 1%, tmpRange: 1%, endStopParent: 3e-3

λ_{10} startStopParent: 100%, tmpRange: 2e-4, maximumBounds: 3e-5

λ_{11} startStopParent: 100%, styleTagId: 2e-3, endStopParent: 1e-3

λ_{12} tmpRange: 99%, selection: 9e-3, startStopParent: 2e-3

λ_{13} startStopParent: 96%, tmpRange: 2%, endStopParent: 1%

λ_{14} tmpRange: 98%, selection: 1%, maximumBounds: 1%

λ_{15} tmpRange: 98%, selection: 2%, maximumBounds: 4e-3

λ_{16} startStopParent: 43%, styleTagId: 29%, endStopParent: 21%

λ_{17} tmpRange: 70%, selection: 14%, maximumBounds: 8%

λ_{18} styleTagId: 84%, tmpRange: 5%, selection: 5%

λ_{19} startStopParent: 98%, endStopParent: 1%, styleTagId: 9e-3

λ_{20} deleteParentBlock: 90%, startStopParent: 4%, selection: 3%

Sample 5

```
...
public static void GetImageFormat(string srcFileName λ1, out string extension λ2,
                                  out ImageFormat imageFormat λ3)
{
    extension λ4 = Path.GetExtension(srcFileName λ5)
                                              .ToLower(CultureInfo.InvariantCulture);
    if (extension λ6 == ".jpg" || extension λ7 == ".jpeg")
    {
        imageFormat λ8 = ImageFormat.Jpeg;
        extension λ9 = ".jpg";
    }
    else if (extension λ10 == ".gif")
    {
        imageFormat λ11 = ImageFormat.Gif;
    }
    else
    {
        imageFormat λ12 = ImageFormat.Png;
        extension λ13 = ".png";
    }
}
...
```

λ_4 extension: 64%, imageFormat: 36%, JPEG_QUALITY: 1e-4

λ_5 extension: 98%, srcFileName: 1%, imageFormat: 1e-3

λ_6 extension: 97%, imageFormat: 1%, srcFileName: 3e-4

λ_7 extension: 75%, JPG: 4%, GIF: 4%

λ_8 imageFormat: 100%, extension: 1e-5, JPEG_QUALITY: 2e-6

λ_9 extension: 93%, imageFormat: 2%, JPEG: 9e-3

λ_{10} extension: 52%, imageFormat: 15%, ICO: 6%, JPG: 6%, GIF: 6%

λ_{11} imageFormat: 100%, extension: 4e-4, JPEG_QUALITY: 1e-5

λ_{12} imageFormat: 99%, JPEG_QUALITY: 4e-3, extension: 2e-3

λ_{13} extension: 66%, JPG: 6%, ICO: 6%, GIF: 6%, BMP: 6%

Sample 6

```
...
BitmapData destBitmapData λ1 = scaledBitmap λ2.LockBits(
        new Rectangle(0, 0, destWidth λ3, destHeight λ4),
        ImageLockMode.WriteOnly, scaledBitmap λ5.PixelFormat);
try
{
    byte* s0 λ6 = (byte*)sourceBitmapData λ7.Scan0.ToPointer();
    int sourceStride λ8 = sourceBitmapData λ9.Stride;
    byte* d0 λ10 = (byte*)destBitmapData λ11.Scan0.ToPointer();
    int destStride λ12 = destBitmapData λ13.Stride;

    for (int y λ14 = 0; y λ15 < destHeight λ16; y λ17++)
    {
        byte* d λ18 = d0 λ19 + y λ20 * destStride λ21;
        byte* sRow λ22 = s0 λ23 + ((int)(y λ24 * yRatio λ25)
                                + yOffset λ26) * sourceStride λ27 + xOffset λ28;
...
```

λ_7 sourceBitmapData: 72%, destBitmapData: 28%, bitmap: 2e-6

λ_9 sourceBitmapData: 90%, destBitmapData: 10%, bitmap: 1e-4

λ_{11} sourceBitmapData: 75%, destBitmapData: 25%, s0: 1e-5

λ_{13} sourceBitmapData: 83%, destBitmapData: 17%, bitmap: 3e-4

Sample 7

```
...
private static Stream GetStreamForUrl(string url λ1, string pageUrl λ2,
                                                    IHTMLElement element λ3)
{
    if (UrlHelper.IsFileUrl(url λ4))
    {
        string path λ5 = new Uri(url λ6).LocalPath;
        if (File.Exists(path λ7))
        {
            return File.OpenRead(path λ8);
        }
        else
        {
            if (ApplicationDiagnostics.AutomationMode)
                Trace.WriteLine("File " + url λ9 + " not found");
            else
                Trace.Fail("File " + url λ10 + " not found");
            return null;
        }
    }
    else if (UrlHelper.IsUrlDownloadable(url λ11))
    {
        return HttpRequestHelper.SafeDownloadFile(url λ12);
    }
    else
    {
...
```

λ_6 url: 96%, element: 2%, pageUrl: 1%

λ_7 path: 86%, url: 14%, element: 1e-3

λ_8 path: 99%, url: 1%, pageUrl: 4e-5

λ_9 path: 97%, url: 2%, pagrUrl: 4e-3%

λ_{10} path: 67%, url: 24%, pageUrl: 5%

Sample 8

```
...
public static void ApplyAlphaShift(Bitmap bitmap λ1, double alphaPercentage λ2)
{
    for (int y λ3 = 0; y λ4 < bitmap λ5.Height; y λ6++)
    {
        for (int x λ7 = 0; x λ8 < bitmap λ9.Width; x λ10++)
        {
            Color c λ11 = bitmap.GetPixel(x λ12, y λ13);
            if (c λ14.A > 0) //never make transparent pixels non-transparent
            {
                int newAlphaValue λ15 = (int)(c λ16.A * alphaPercentage λ17);
                //value must be between 0 and 255
                newAlphaValue λ18 = Math.Max(0, Math.Min(255, newAlphaValue λ19));
                bitmap λ20.SetPixel(x λ21, y λ22, Color.FromArgb(newAlphaValue λ23, c λ24));
            }
            else
                bitmap λ25.SetPixel(x λ26, y λ27, c λ28);
        }
    }
}
...
```

λ14 alphaPercentage: 52%, bitmap: 32%, c: 13%

λ16 bitmap: 67%, alphaPercentage: 27%, c: 4%

λ17 alphaPercentage: 85%, c: 6%, JPEQ_QUALITY: 3%

λ18 newAlphaValue: 51%, bitmap: 24%, alphaPercentage: 11%

λ19 newAlphaValue: 86%, y: 4%, alphaPercentage: 3%

λ20 bitmap: 100%, c: 4e-3, JPEG_QUALITY: 3e-4

λ21 bitmap: 98%, x: 9e-2, c: 8e-3

λ22 c: 50%, bitmap: 49%, newAlphaValue: 2e-3, y: 3e-8

λ23 alphaPercentage: 42%, JPEG_QUALITY: 40%, bitmap: 10%, newAlphaValue: 3%

λ24 newAlphaValue: 60%, alphaPercentage: 25%, c: 5%

λ25 bitmap: 100%, c: 8e-4, alphaPercentage: 3e-4

λ26 bitmap: 88%, x: 9%, c: 2%

λ27 c: 79%, bitmap: 18%, JPEG_QUALITY: 1%, y: 3e-3

λ28 c: 82%, y: 6%, x: 4%

Sample 9

```
...
string s[λ1] = (string)Value[λ2];
byte[] data[λ3];

Guid g[λ4];
if (s[λ5].Length == 0)
{
    data[λ6] = CollectionUtils.ArrayEmpty<byte>();
}
else if (ConvertUtils.TryConvertGuid(s[λ7], out g[λ8]))
{
    data[λ9] = g[λ10].ToByteArray();
}
else
{
    data[λ11] = Convert.FromBase64String(s[λ12]);
}

SetToken(JsonToken.Bytes, data[λ13], false);
return data[λ15];
...
```

λ_2 t: 58%, Value: 12%, _tokenType: %

λ_5 TokenType: 44%, QuoteChar: 43%, _currentPosition: 4%, s: 9e-3

λ_6 _tokenType: 74%, data: 20%, _currentState: 5%

λ_7 QuoteChar: 31%, ValueType: 26%, Path: 9%, s: 3e-4

λ_8 g: 100%, data: 6e-5, t: 3e-5

λ_9 data: 99%, _tokenType: 5e-3, ValueType: 9e-4

λ_{10} g: 99%, data: 6e-3, _currentState: 2e-3

λ_{11} data: 66%, _tokenType: 31%, _currentState: 6e-3

λ_{12} s: 74%, Value: 20%, t: 3%

D Dataset

The collected dataset and its characteristics are listed in Table 2.

Table 2: Projects in our dataset. Ordered alphabetically. kLOC measures the number of non-empty lines of C# code. Projects marked with ^{Dev} were used for validation. Projects marked with † were in the test-only dataset. The rest of the projects were split into train-validation-test. The dataset contains in total about 4,824kLOC.

Name	Git SHA	kLOCs	plhldrs	Description
Akka.NET	`9e76d8c`	236	183k	Actor-based Concurrent & Distributed Framework
AutoMapper	`6dd6adf`	43	21k	Object-to-Object Mapping Library
BenchmarkDotNet†	`b4d68e9`	23	1k	Benchmarking Library
BotBuilder†	`a6be5de`	43	32k	SDK for Building Bots
choco	`73b7035`	34	21k	Windows Package Manager
CommonMark.NET^{Dev}	`e94800e`	14	7k	Markdown Parser
Dapper	`637158f`	18	1k	Object Mapper Library
EntityFramework	`fa0b7ec`	263	184k	Object-Relational Mapper
Hangfire†	`ffc4912`	33	32k	Background Job Processing Library
Nancy	`422f4b4`	69	49k	HTTP Service Framework
Newtonsoft.Json	`744be1a`	119	70k	JSON Library
Ninject	`dbb159b`	13	3k	Code Injection Library
NLog	`3954157`	67	37k	Logging Library
OpenLiveWriter	`78d28eb`	290	159k	Text Editing Application
Opserver	`c0b70cb`	24	16k	Monitoring System
OptiKey	`611b94a`	27	14k	Assistive On-Screen Keyboard
orleans	`eaba323`	223	133k	Distributed Virtual Actor Model
Polly	`b5446f6`	30	25k	Resilience & Transient Fault Handling Library
ravendb^{Dev}	`2258b2c`	647	343k	Document Database
RestSharp	`e7c65df`	20	20k	REST and HTTP API Client Library
roslyn	`d18aa15`	1,997	1,034k	Compiler & Code Analysis
Rx.NET	`594d3ee`	180	67k	Reactive Language Extensions
scriptcs†	`ca9f4da`	18	14k	C# Text Editor
ServiceStack	`b0aacff`	205	33k	Web Framework
ShareX	`52bcb52`	125	91k	Sharing Application
SignalR	`fa88089`	53	35k	Push Notification Framework
Wox†	`cdaf627`	13	7k	Application Launcher