Microsoft Fast and Memory-Efficient Neural Code Completion

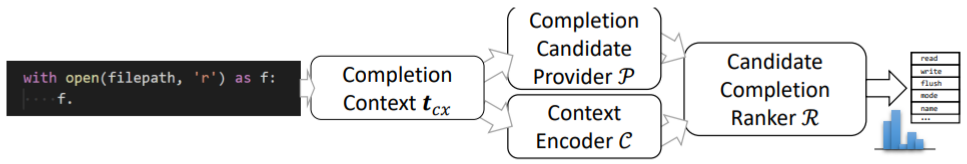
2787612782

0604 2020

1 Problems/Backgrounds

problems: state-of-the-art neural network models consume prohibitively large amounts of memory, causing computational burden to the development environment, especially when deployed in lightweight client devices.
Methods: we reframe neural code completion from a generation task to a task of learning to rank the valid completion suggestions computed from static analyses.

2 Methods/Techonologies



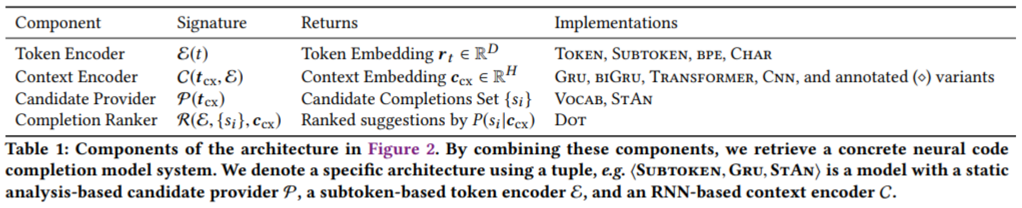
```
with open(filepath, 'r') as f:
    f.
```

Completion Context t_{cx}
Completion Candidate Provider $\mathcal{P}$
Context Encoder $\mathcal{C}$
Candidate Completion Ranker $\mathcal{R}$
read
write
flush
mode
name
...

Figure 2: Our architecture for machine learning-based code completion systems. From the developer environment, the (partial) code context t_{cx} is extracted and passed into the context encoder, $\mathcal{C}$, that "summarizes" the completion context. Simultaneously, a candidate provider $\mathcal{P}$ receives the code context and computes a set of candidate completions for the current completion location. Finally, a completion ranker $\mathcal{R}$ ranks the candidates given the summarized context and presents them to the user. In this work, we instantiate each component with various types of neural networks (Table 1), showing how different choices affect suggestion accuracy, model size and computational efficiency. All our neural models are trained end-to-end.

Figure 1: The architecture

Token Encoder A neural network E that encodes a code token t into a distributed vector representation (embedding) rt .
Context Encoder A neural network C that encodes (i.e. summarizes) the completion context t cx, into a distributed vector representation (embedding) ccx.
Candidate Provider P A component that accepts the completion context t cx and yields an (unordered) set of M candidate completion targets si , i.e. P(t



Component	Signature	Returns	Implementations
Token Encoder	$\mathcal{E}(t)$	Token Embedding $\boldsymbol{r}_t \in \mathbb{R}^D$	Token, Subtoken, bpe, Char
Context Encoder	$C(\boldsymbol{t}_{\text{cx}}, \mathcal{E})$	Context Embedding $\boldsymbol{c}_{\text{cx}} \in \mathbb{R}^H$	Gru, biGru, Transformer, Cnn, and annotated (⋄) variants
Candidate Provider	$\mathcal{P}(\boldsymbol{t}_{\text{cx}})$	Candidate Completions Set $\{s_i\}$	Vocab, StAn
Completion Ranker	$\mathcal{R}(\mathcal{E}, \{s_i\}, \boldsymbol{c}_{\text{cx}})$	Ranked suggestions by $P(s_i \| \boldsymbol{c}_{\text{cx}})$	Dot

Table 1: Components of the architecture in Figure 2. By combining these components, we retrieve a concrete neural code completion model system. We denote a specific architecture using a tuple, *e.g.* ⟨Subtoken, Gru, StAn⟩ is a model with a static analysis-based candidate provider $\mathcal{P}$, a subtoken-based token encoder $\mathcal{E}$, and an RNN-based context encoder C.

Figure 2: The comonents

cx) = si = s0, ...,sM
Completion RankerA neural component R that accepts the context encoding ccx, along with a set of candidate completion targets si , and ranks them.

3 Validation/Results

we focus on a particular instance of code completion: the completion of method invocations and field accesses (i.e. API completion).
Datasets:
The training, validation and test data used for the experiments came from the 2700 top-starred Python source code repositories on GitHub. Scraped dataset may contain a large amount of duplicates [1, 31], which may skew the evaluation results. To avoid this, we deduplicate our dataset using the tool of Allamanis [1]. Our dataset contains libraries from a diverse set of domains including scientific computing, machine learning, dataflow programming, and web development. To prepare the data, we use the type inference engine currently in Visual Studio Code (PTVS) to collect all completion locations where an API completion suggestion would be emitted by PTVS, similar to Svyatkovskiy et al. [43]. We extract the previous N = 80 tokens preceding the method invocation (tcx), and information about the receiver object or namespace, which we use to recreate a Python StAn candidate completion provider. The final dataset contains approximately 255k files and a total of 7.4 million API completion instances. We split the data per-file into training-validation-test at 60-20-20 proportions.
Metrics
We measure three aspects of the completion models: **accuracy**:mean reciprocal rank (MRR) ,Recall@k **model size** :compute the total number of parameters of each model **suggestion speed**:we compute the average time needed for our neural network to compute a single suggestion on a CPU
Baseline
We compare our models to a simple baseline (“Popularity”) that yields the most frequently used target completion for a given API in our training set.
Results

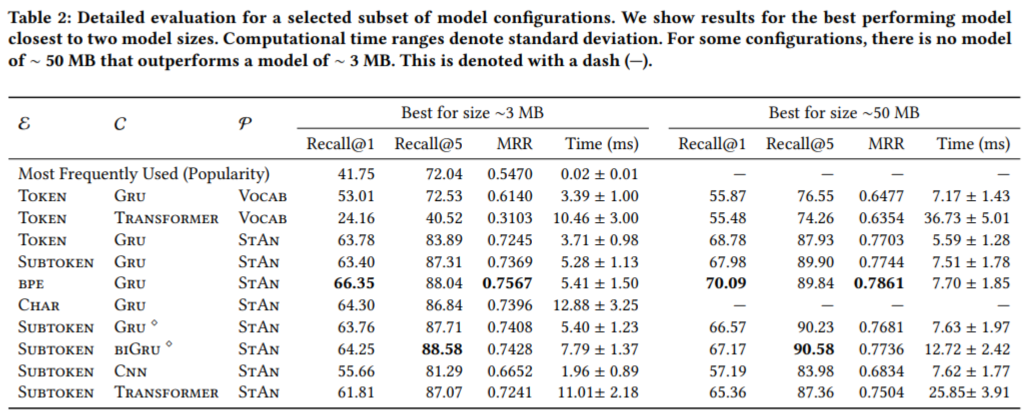


Table 2: Detailed evaluation for a selected subset of model configurations. We show results for the best performing model closest to two model sizes. Computational time ranges denote standard deviation. For some configurations, there is no model of ~ 50 MB that outperforms a model of ~ 3 MB. This is denoted with a dash (—).

$\mathcal{E}$	$\mathcal{C}$	$\mathcal{P}$	Best for size ~3 MB				Best for size ~50 MB			
			Recall@1	Recall@5	MRR	Time (ms)	Recall@1	Recall@5	MRR	Time (ms)
Most Frequently Used (Popularity)			41.75	72.04	0.5470	0.02 ± 0.01	—	—	—	—
Token	Gru	Vocab	53.01	72.53	0.6140	3.39 ± 1.00	55.87	76.55	0.6477	7.17 ± 1.43
Token	Transformer	Vocab	24.16	40.52	0.3103	10.46 ± 3.00	55.48	74.26	0.6354	36.73 ± 5.01
Token	Gru	StAn	63.78	83.89	0.7245	3.71 ± 0.98	68.78	87.93	0.7703	5.59 ± 1.28
Subtoken	Gru	StAn	63.40	87.31	0.7369	5.28 ± 1.13	67.98	89.90	0.7744	7.51 ± 1.78
bpe	Gru	StAn	**66.35**	88.04	**0.7567**	5.41 ± 1.50	**70.09**	89.84	**0.7861**	7.70 ± 1.85
Char	Gru	StAn	64.30	86.84	0.7396	12.88 ± 3.25	—	—	—	—
Subtoken	Gru ◇	StAn	63.76	87.71	0.7408	5.40 ± 1.23	66.57	90.23	0.7681	7.63 ± 1.97
Subtoken	biGru ◇	StAn	64.25	**88.58**	0.7428	7.79 ± 1.37	67.17	**90.58**	0.7736	12.72 ± 2.42
Subtoken	Cnn	StAn	55.66	81.29	0.6652	1.96 ± 0.89	57.19	83.98	0.6834	7.62 ± 1.77
Subtoken	Transformer	StAn	61.81	87.07	0.7241	11.01± 2.18	65.36	87.36	0.7504	25.85± 3.91

Figure 3: The results