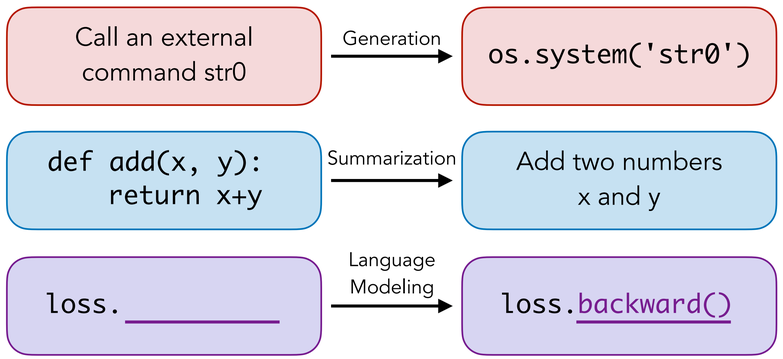
Nearest Neighbor Code Generation and Summarization

Alex Xie, mentored by Vincent Hellendoorn
07-400 Spring 2022

Carnegie Mellon University

Introduction

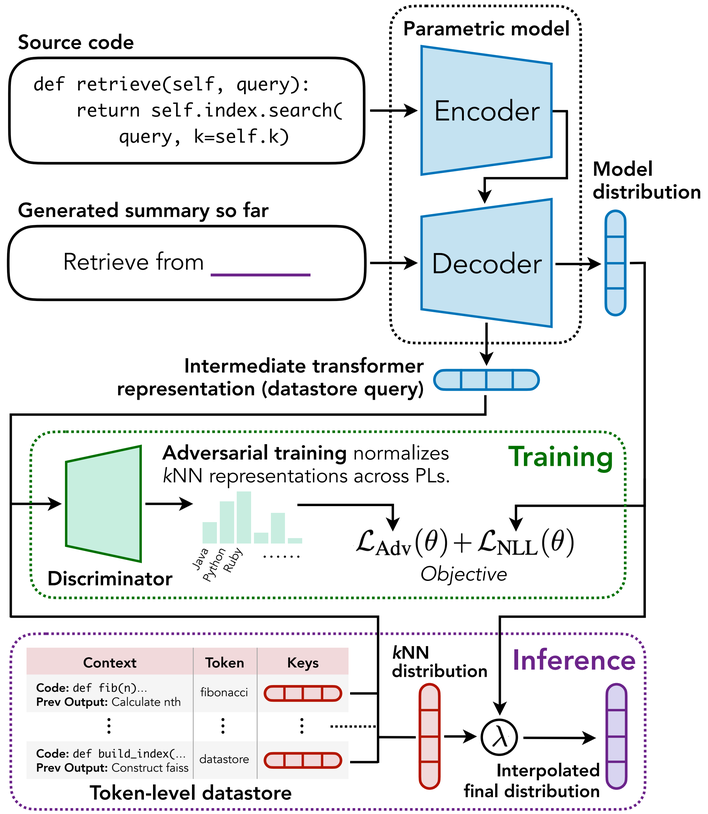


- We study the utility of **neural probabilistic retrieval** on three code-related sequence generation tasks:
  - **Code generation**: Given a natural language specification, generate an appropriate function/code snippet.
  - **Code summarization**: Given a piece of source code, generate a brief natural language summary.
  - **Code language modeling**: Estimate the distribution of sequences of source code tokens; equivalently, predict the next token given a piece of code as context.

Motivation

- Like natural lanugage, source code contains many long-tailed patterns; we hypothesize that it may help to explicitly **memorize rare patterns via retrieval** rather than relying solely upon the model's parameters.
- Retrieval has been shown to aid **high-resource to low-resource transfer** across natural languages; we investigate whether this holds analogously for programming languages.
- Retrieval-based methods have also been found to enable **efficient domain adaptation with minimal retraining**. We explore the idea of retrieval-guided zero-shot and few-shot adaptation to previously unseen programming languages.

Method



We augment **large code language models** (CodeT5, GPT-2) with a token-level neural retrieval mechanism (**kNN-LM**), allowing them to leverage a **large external knowledge base** when generating code or summaries.

Future Work

- Scaling up our approach to (i) **larger models** (i.e. billion-scale models such as Polycoder & CodeGen) and (ii) **larger datastores**.
- Incorporating more abundant **unpaired data** into code summarization and code generation datastores via back-translation.

Experiments

Code Summarization

On CodeSearchNet, a dataset of 800k+ function-comment pairs across six PLs, we find consistent BLEU score improvements from kNN retrieval.

	Ruby	JS	Go	Pyth.	Java	PHP
Base	16.01	16.24	19.74	19.43	20.29	26.23
+ kNN	16.04	**17.92**	19.46	19.46	**20.73**	**26.59**
+ kNN + Adv	**16.22**	16.42	**19.77**	**19.52**	20.42	26.50

We further find that Ruby, the lowest-resource language, benefits from retrieving from higher-resource languages (i.e. Python → En., Java → En.).

No kNN	Ruby-only	+Python	+Java	+All
16.01	16.04	16.24	**16.31**	16.30

Code Generation

We find kNN underperforms for code generation, yielding marginal improvements over the base model; we show a success case and a failure case below.

Intent	get the integer location of a key var0 in a pandas data frame	
GT	`df.index.get_loc('var0')`	
Base	`df['var0'].apply(lambda x: x.index.get_loc(var0))`	✗
kNN	`df.index.get_loc(var0)`	✓
Intent	convert a raw string var0 into a normal string	
GT	`var0.decode('string_escape')`	
Base	`var0.decode('string_escape')`	✓
kNN	`unicodedata.normalize('NFKD', var0).encode('ascii', 'ignore')`	✗

Code Language Modeling

We find that kNN-LM aids with language adaptation (in this case to Ruby), though it is often not effective as training for another 50-100 steps. Notably however, retrieval yields improvements even in a zero-shot setting (i.e. at step 0), which may be of practical use when adapting large language models.