Selective Retrieval-Augmented Infilling for Repository-Level Code Completion

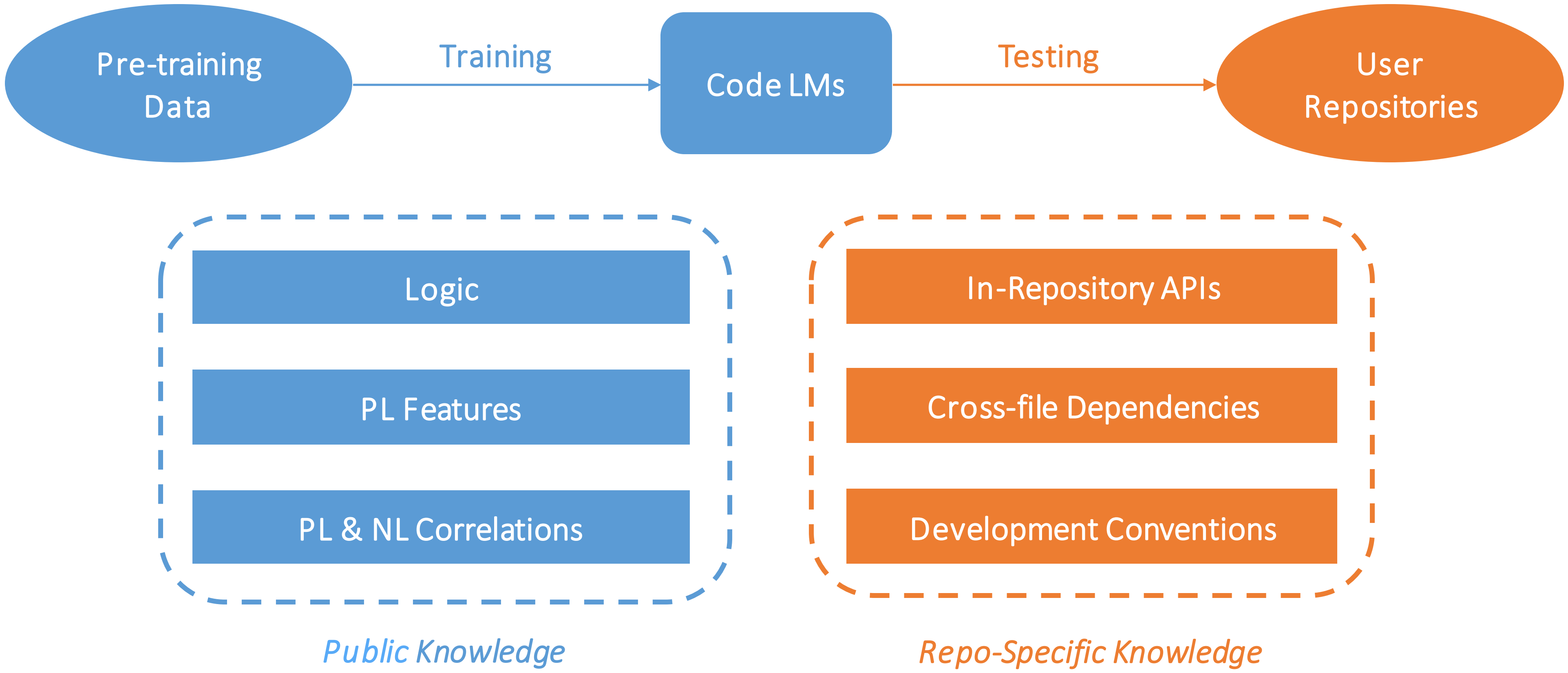
Di Wu, Wasi Ahmad, Dejiao Zhang

2023/10/16

A million-dollar question

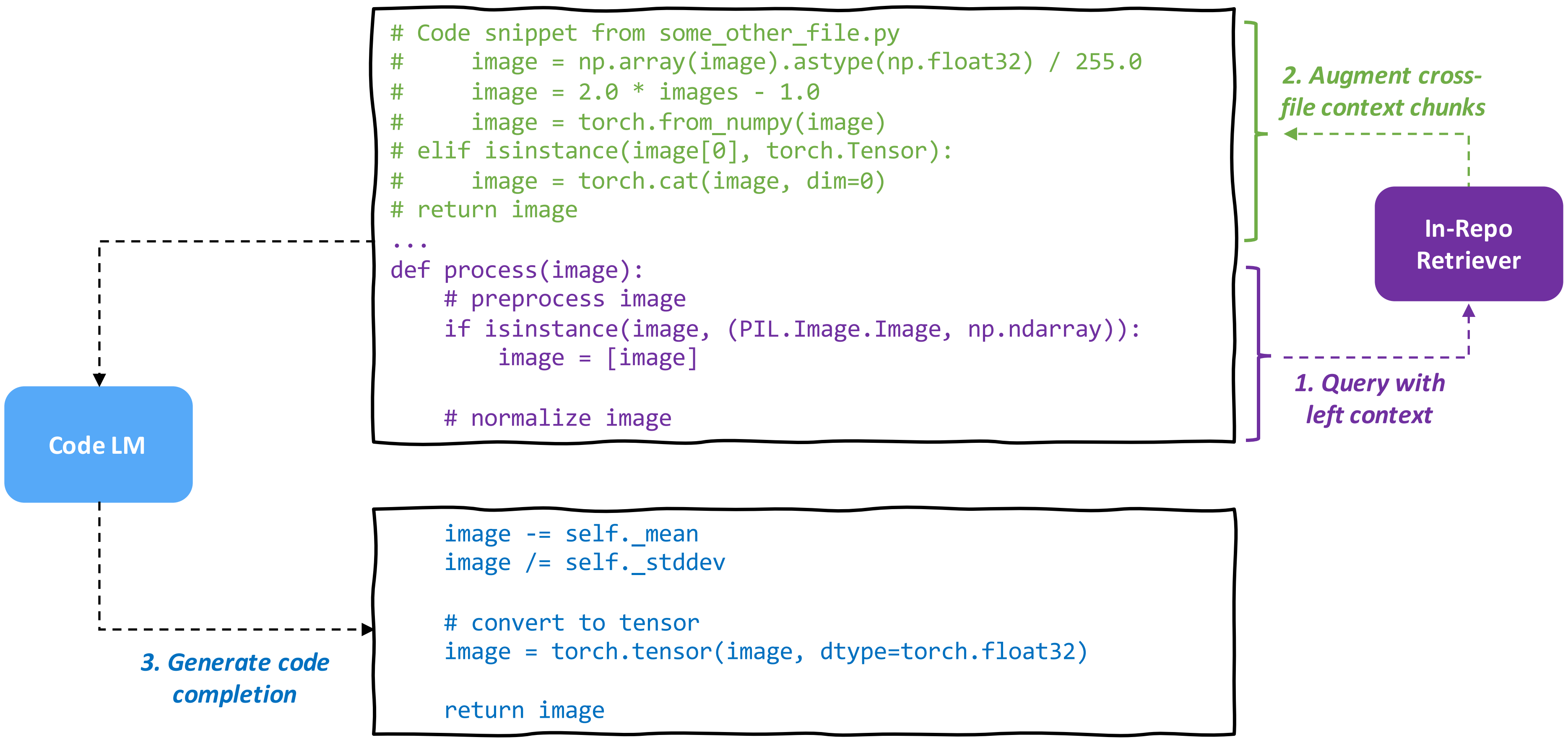
- ***How to fill in a hole in an arbitrary repository?***
- Code language models (LMs) have shown promising performance.

Challenge: the knowledge gap



Retrieval-Augmented Generation (RAG)

- A successful system (RepoCoder, EMNLP 2023)



Improving the paradigm

- Issue: existing works treat ***right contexts*** as cross-file information.
  - Failure to capture the code immediately following the hole.
  - Fixed-size chunks may fail to capture the entire set of useful information.
  - Many LMs are already trained on fill-in-the-middle, e.g., StarCoder [1].

- We propose directly give both left and right contexts in the prompt.

[1] StarCoder: may the source be with you! Li et al., arXiv 2023.

Improving the paradigm

- We propose directly providing both left and right contexts in the prompt.

```
# prompt for CodeGen [1]
[CFC] RC LC
```

```
# prompt for StarCoder [2]
<fim_prefix> [CFC] LC <fim_suffix> RC <fim_middle>
```

* LC = left context, RC = right context, CFC = retrieved cross-file context chunks

[1] CodeGen: An Open Large Language Model for Code with Multi-Turn Program Synthesis, Nijkamp et al., ICLR 2023.
[2] StarCoder: may the source be with you! Li et al., arXiv 2023.

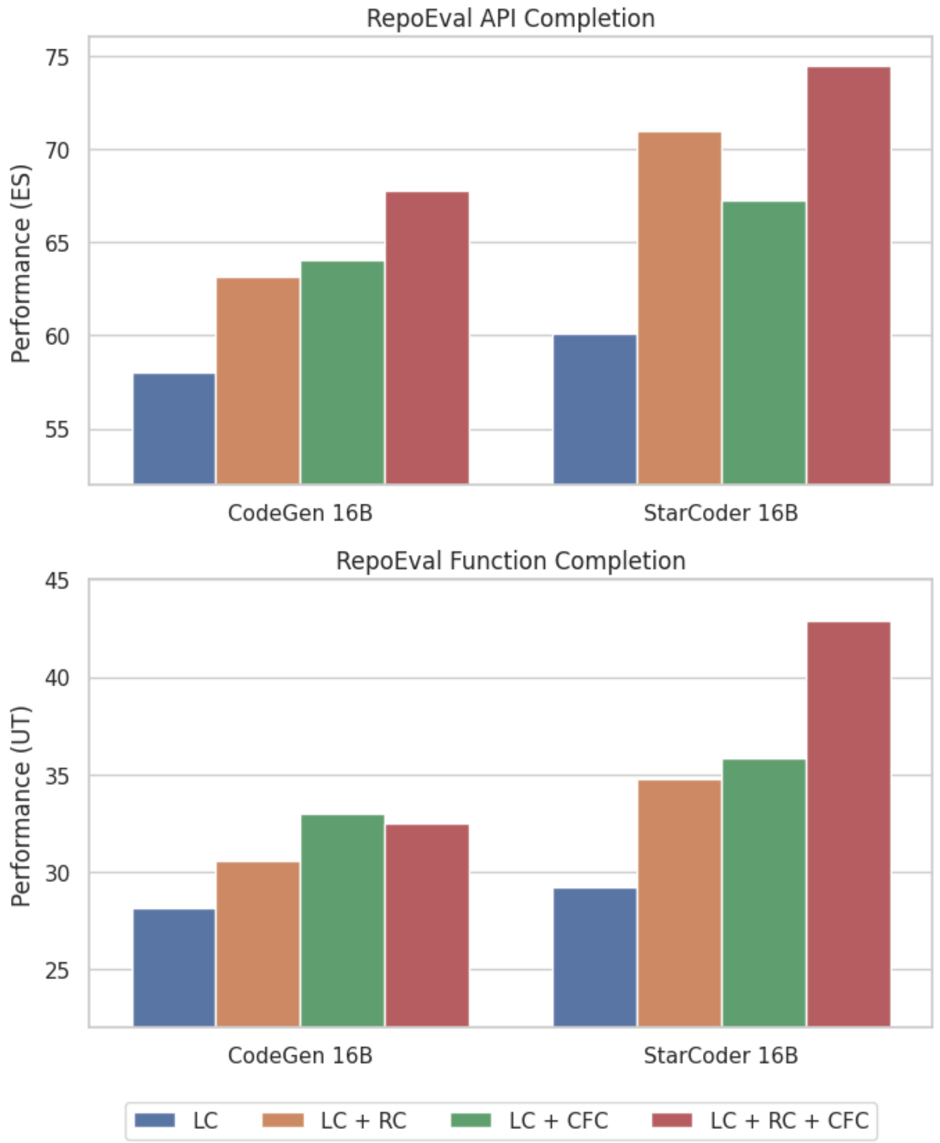
Evaluation

- Repo-level code generation tasks from RepoEval [1]:
  - Line completion
  - API completion
  - Function completion
- Metrics
  - Exact match (EM, upper bound for correctness)
  - Edit similarity (ES, user experience)
  - Unit test pass rate (UT, correctness of function completion)

[1] RepoCoder: Repository-Level Code Completion Through Iterative Retrieval and Generation, Zhang et al., EMNLP 2023.

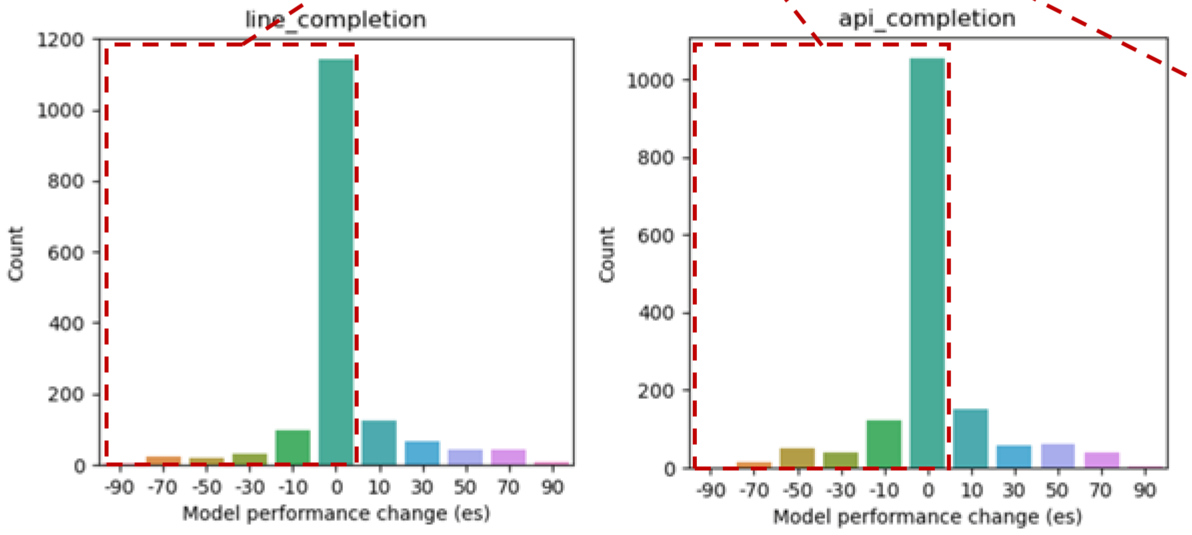
Results

- Providing RC generally improves the completion performance.
- StarCoder, pre-trained on FIM, is better at leveraging the RC.
- We will focus on the ***Retrieval-Augmented Infilling (RAI)*** setup with StarCoder.



The 80-20 rule for RAI

- Is retrieval beneficial for every instance?
- We find 80% of the retrievals could be avoided with no performance loss.



function_completion

Fail -> Pass	No Change	Pass -> Fail
53	385	17

Model performance change (ut)

Selective Retrieval-Augmented Infilling

- Since the gain from retrieval is sparse, it is important to understand:
  - ***When to retrieve?***
  - ***How to maximally leverage the retrieved context?***

- Therefore, we formulate the novel task of ***Selective RAI***.

- Always decide whether CFC is required for the infilling task.
  - No → directly use (LC, RC) to prompt the code LM.
  - Yes → retrieve CFC and prompt the LM with (LC, RC, CFC)

Evaluating Selective RAI

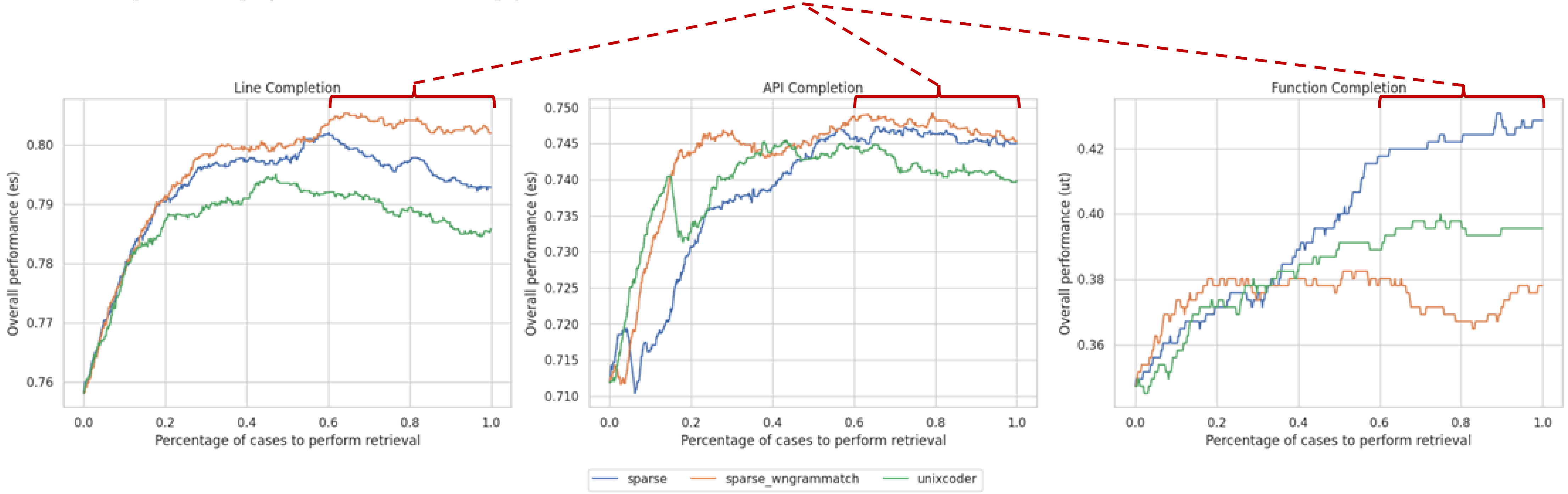
- Selective RAI system are evaluated according to the two questions

- ***The performance-budget trade-off***
  - A superior system should achieve the same level of performance with less retrieval budget.

- ***Ratio of performance gain and loss on the retrieval instances***
  - A superior system should exhibit performance improvement on all the instances where it decides to retrieve.

Leveraging Retrievers to solve Selective RAI

- A naïve baseline: use the retriever's similarity to make selections.
- We select top k% instances to perform retrieval-augmented infilling, while performing in-file infilling for the rest (100-k)% instances.
- Surprisingly, this strategy saves at least 40% retrievals on StarCoder 16B.



Limitations

- Practical considerations
  - Finding a proper similarity ***threshold*** could be challenging in practice.
  - ***Retrieval is required*** to calculate the similarity score, which is expensive.
- Performance considerations
  - Ignores the case where the model already makes good predictions without CFC.
  - Prompts with CFCs are OOD for code LMs, possibly harming the performance.
- Therefore, we must also ***adapt the LM itself*** to better solve Selective RAI.

Adapting Code LMs for Selective RAI

- Our problems at hand:
  - How to utilize the information from the LM side for S-RAI?
  - How to avoid the negative effects of the retrieved context in S-RAI systems?
  - How to avoid performing the retrieval before making the selective decision?

- Our proposal: ***self-triggered retrieval***
  - *Let the LM selectively request for the CFC after observing the in-file context.*

Adapting Code LMs for Selective RAI

- *selectively request for the CFC after observing the in-file context?*
- Our insight: this is a form of *self-planning, or self-evaluation.*

<fim_prefix> left_context <fim_suffix> right_context <fim_middle> ?

Observed Context

Do I know the answer here?

Can I answer it given more CFC?

- Training a calibrated LM to self-evaluate is viable and investigated by prior work [1].
- For our task, the ground truth can be easily labelled.

[1] Language Models (Mostly) Know What They Know, Kadavath et al., arXiv 2022.

Infilling with Self-Trigged Retrieval

- Two new tokens: <end_suffix> and <cfc_info>

Self-assessment | Provided only on request

<fim_prefix> left_context <fim_suffix> right_context <end_suffix> <cfc_info> CFC <fim_middle> completion

Prompt

- The model *self-evaluates* whether it needs extra context for better infilling.
  - If so, it predicts <cfc_info>, and we provide CFC ending with <fim_middle>.
  - If not, we directly append <fim_middle>.
- One relaxation: we use the probability of <cfc_info> as the decision criteria.

Infilling with Self-Trigged Retrieval

- Training

Self-assessment | Provided only on request

<fim_prefix> left_context <fim_suffix> right_context <end_suffix> <cfc_info> CFC <fim_middle> completion

Prompt

- A multi-task objective
  - *Self-assessment loss*: Pr(<cfc_info> | prompt)
  - *Code completion loss*: Pr(completion | prompt + optional CFC)
  - We do not supervise the prompt, CFC tokens, or <fim_middle>

Infilling with Self-Trigged Retrieval

- Training data creation process (simplified)
  1. Sample a hole to fill in and record the ground truth and the in-file context.
  2. Run repo-level retrieval and record the top-3 relevant code chunks as the CFC.
  3. Run inference with a code LM *twice*

     ```
     <fim_prefix> left_context <fim_suffix> right_context <fim middle>      → completion_in_file
     <fim_prefix> left_context <fim_suffix> right_context CFC <fim middle>  → completion_with_cfc
     ```

  4. Label via edit similarity evaluation

     Label ← `ES(ground truth, completion_in_file) < ES(ground truth, completion_with_cfc)`

- If label = True, train on (1) requesting for retrieval and (2) retrieval-augmented infilling.
- Otherwise, train on (1) not requesting for retrieval, and (2) infilling without retrieval.

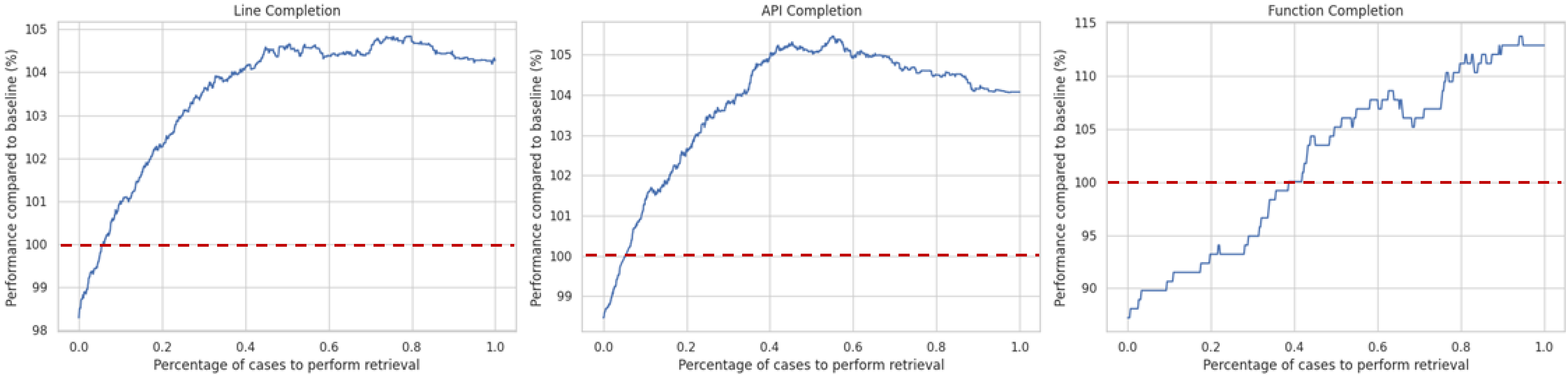
Infilling with Self-Trigged Retrieval

- Advantages
    - Self-triggered retrieval allows a model to smoothly self-switch between RAI and infilling.
        - Learning self-evaluation without losing generality.
        - In addition, fine-tuning on RAI to avoid negative retrieval.
        - No extra latency if retrieval is not triggered.
    - Our paradigm exploits existing data in a self-supervised manner, with low labeling costs.

- More training details
    - We create 350k chunk and function completion instances using 20k repos.
    - We adapt StarCoderBase-1B/3B models and call them ***Repoformer-1B/3B***.
    - The two losses are assigned equal weights.
    - 2 epochs with LR 1e-5, BSZ 512, 100 warmup steps, and linear LR decay.

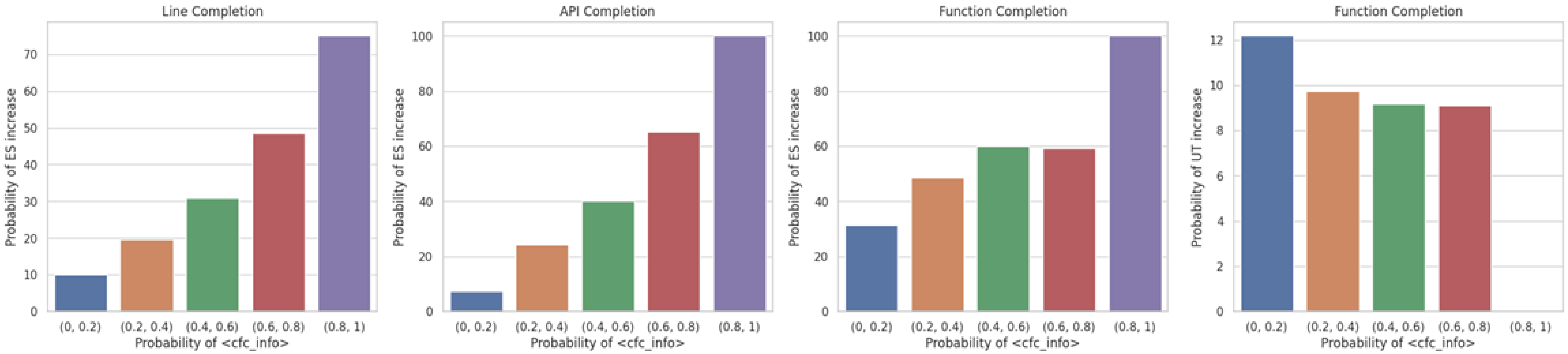
Repoformer-1B Evaluations

- Baseline: prompting StarCoderBase-1B with left, right, and cross-file context.
- Self-selecting cases for RAI, **Repoformer-1B outperforms the baseline with very small retrieval budget.**
  - ~8% for line/API completion, ~40% for function completion.
- ~5% overall performance gain for line/API completion and ~13% gain for function completion.



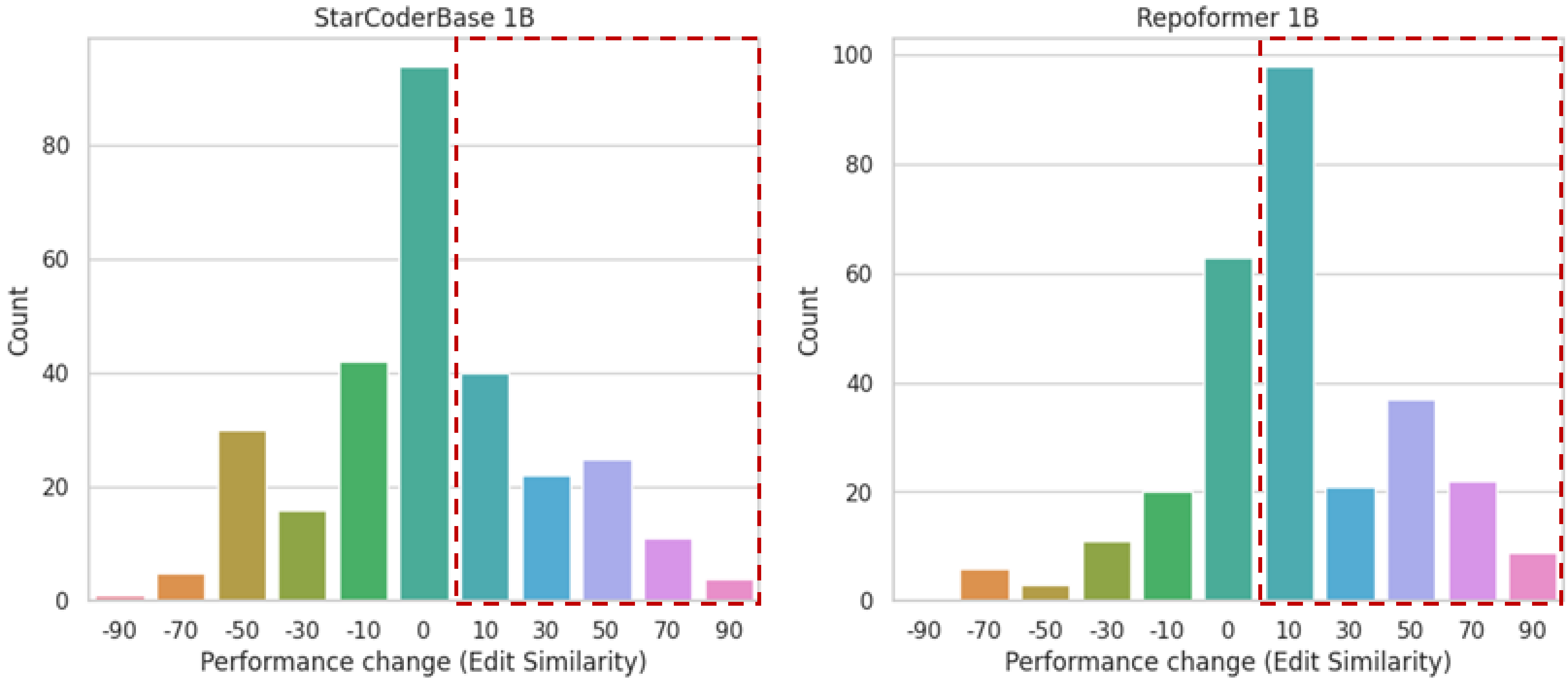
Repoformer-1B Evaluations

- **Repoformer makes roughly-calibrated decisions for retrieval but is often over-confident.**
- Probability of ES increase – calculated by prompting the model twice.
- Limitation: Repoformer cannot predict the gain in UT pass rate very well.



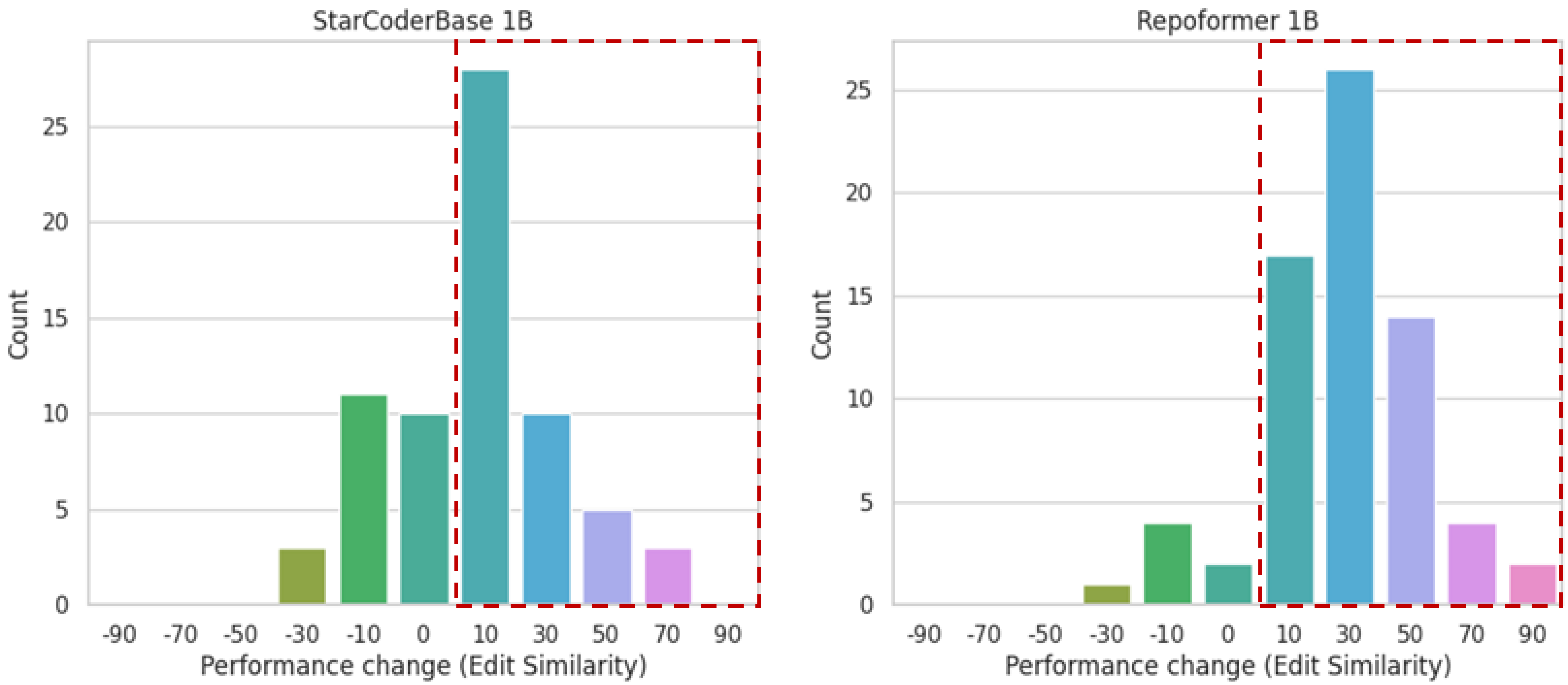
Repoformer-1B Evaluations

- **Repoformer is better at leveraging the retrieved CFCs.**
- We compare the performance gain from CFCs of Repoformer vs. StarCoderBase on the instances self-selected by Repoformer. (RepoEval API Completion)



Repoformer-1B Evaluations

- **Repoformer is better at leveraging the retrieved CFCs.**
- We compare the performance gain from CFCs of Repoformer vs. StarCoderBase on the instances self-selected by Repoformer. (RepoEval Function Completion)



Performance with tuned threshold

- We tune the threshold on a validation dataset and compare the performance.

model	policy	API Completion			Function Completion		
		threshold	% retrieval	ES	threshold	% retrieval	ES
StarCoder 1B	-	-	0%	66.54	-	0%	47.65
	retriever sim	0.622	75%	69.23	0.397	99%	55.71
	-	-	100%	69.17	-	100%	55.64
Repoformer 1B	-	-	0%	68.14	-	0%	50.68
	retriever sim	0.563	88%	72.18	0.110	100%	57.30
	self selection	0.245	**55%**	**72.98**	0.081	**90%**	**57.41**
	-	-	100%	72.02	-	100%	57.30

Limitations & Extensions

- Experiments are only on Python.
- Edit Similarity as the training signal.
- Stronger results could be obtained if the “on-policy” setting is considered by further training Repoformer with RL.
- Repoformer itself can be a planning + drafting tool for much larger code LMs.
- Repository-specific selective policies could be considered.

Discussion

- Our work resonates with many concurrent efforts to make retrieval-augmented and tool-augmented LMs more efficient [1, 2, 3] and robust [4].
  - Perspective 1: selective retrieval as ***extreme context compression*** [1, 2, 3]
  - Perspective 2: selective retrieval as ***single-tool planning*** [5, 6]
  - With proper formulation, a modest-sized LM can be trained as the planner.

- Our method also extends the self-evaluation scheme to a new task [6, 7]
  - We explore embedding simple self-evaluation in language modeling.

[1] RECOMP: Improving Retrieval-Augmented LMs with Compression and Selective Augmentation, Xu et al., arXiv 2023.
[2] Self-Knowledge Guided Retrieval Augmentation for Large Language Models, Wang et al., arXiv 2023.
[3] When Not to Trust Language Models: Investigating Effectiveness of Parametric and Non-Parametric Memories, Mallen et al., ACL 2023.
[4] Making Retrieval-Augmented Language Models Robust to Irrelevant Context, Ran et al., arXiv 2023.
[5] Toolformer: Language Models Can Teach Themselves to Use Tools, Schick et al., arXiv 2023.
[6] Guiding Language Model Reasoning with Planning Tokens, Wang et al., arXiv 2023.
[7] Language Models (Mostly) Know What They Know, Kadavath et al., arXiv 2022.

Summary

- **The 80-20 rule**: retrieval augmentation often does not improve the repository-level code completion performance.
- **The suggestion**: considering ***selective retrieval*** is strongly advised.
- **The solutions**:
    - Retriever’s scores provide useful hints on whether a CFC chunk is useful.
    - Self-supervised adaptation enables LMs to self-trigger retrieval.