ICLR
International Conference On Learning Representations

CODE2JSON: CAN A ZERO-SHOT LLM EXTRACT CODE FEATURES FOR CODE RAG?

Aryan Singhal, Rajat Ghosh, Ria Mundra, Harshil Dadlani, Debojyoti Dutta

{aryan.singhal, rajat.ghosh, debo.dutta}@nutanix.com

NUTANIX

CodeRAG

- Using a pre-trained LLM to answer code-related queries works well for generic code, but fails to perform as well for domain-specific/niche codebases like those used in software engineering tasks.

- Indexing a code-base and using it as a knowledge base of an LLM to answer queries specific to that code is a viable solution, but does not work well due to resource-constrained environments.

- Direct code-indexing yields suboptimal results for natural language queries, die to the relatively low representation of code-data in LLMs.

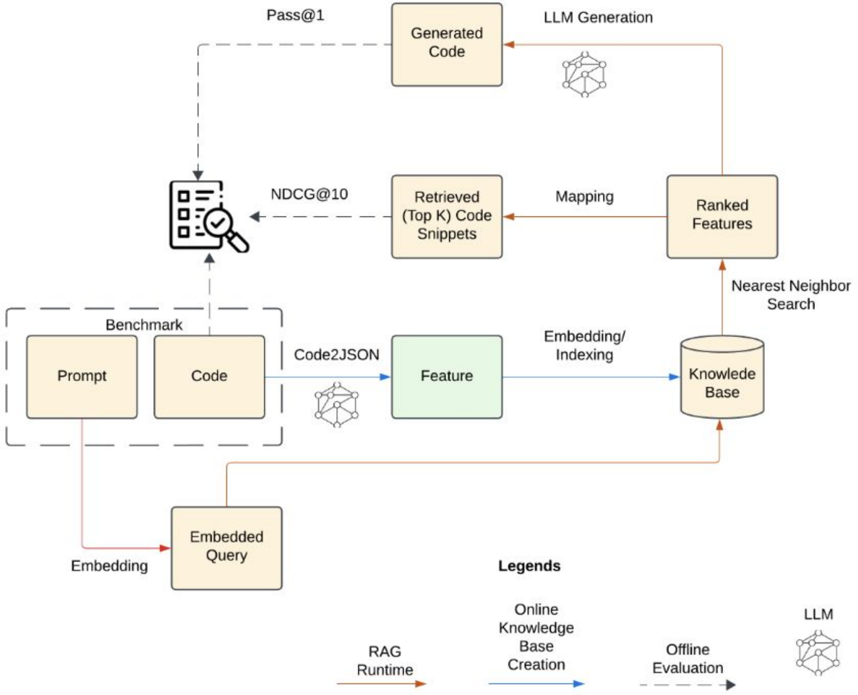
This work introduces Code2JSON, a novel LLM-based framework aimed at extracting meaningful features from code in a zero-shot, generalizable way, useful for both code retrieval and generation tasks.

Code2JSON

- Our work is based on the hypothesis that, a natural language representation of code-data should perform on par, if not better for codeRAG.

- To resolve this, we use an LLM to generate natural language features which are indexed for retrieval.

- These natural language features are hypothesized to enhance retrieval quality and aid in generation as well.



Experiments

- We compare our approach to a baseline approach if simply indexing the code for retrieval.

- Code2JSON shows superior performance to the baseline approach in 4 out of 6 python-native datasets, over different retrieval strategies

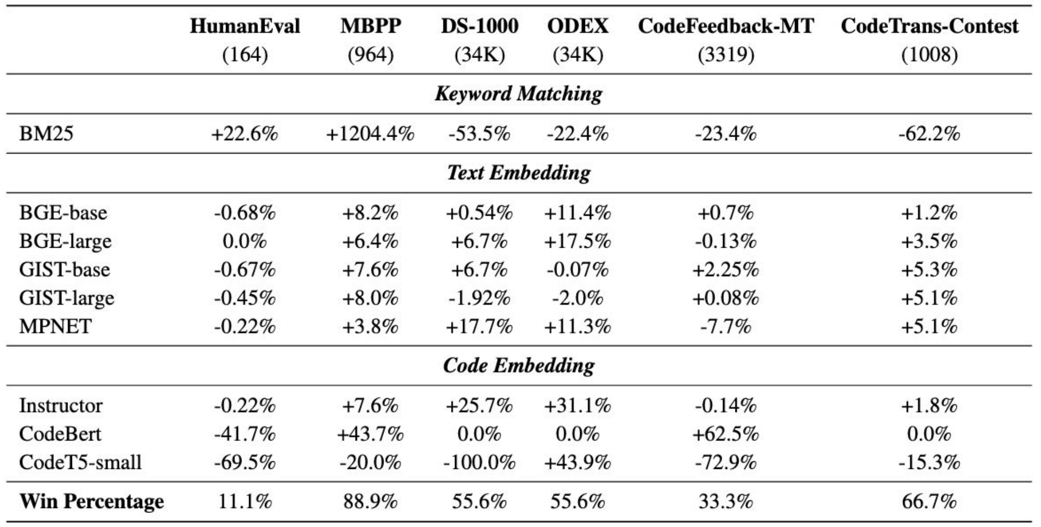


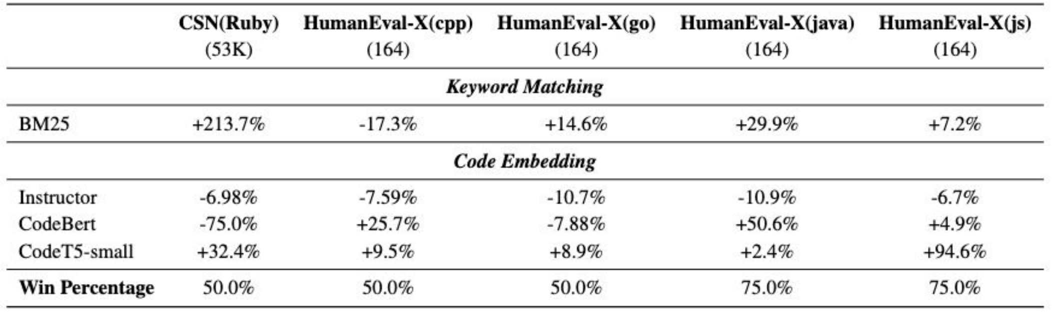
Table 1: Relative improvement in **NDCG@10** from using CODE2JSON feature extraction on Python datasets. Each cell shows the percentage change (positive or negative) relative to indexing raw code. The final row indicates how often the CODE2JSON-based approach outperforms direct code indexing.

	HumanEval (164)	**MBPP** (964)	**DS-1000** (34K)	**ODEX** (34K)	**CodeFeedback-MT** (3319)	**CodeTrans-Contest** (1008)
Keyword Matching						
BM25	+22.6%	+1204.4%	-53.5%	-22.4%	-23.4%	-62.2%
Text Embedding						
BGE-base	-0.68%	+8.2%	+0.54%	+11.4%	+0.7%	+1.2%
BGE-large	0.0%	+6.4%	+6.7%	+17.5%	-0.13%	+3.5%
GIST-base	-0.67%	+7.6%	+6.7%	-0.07%	+2.25%	+5.3%
GIST-large	-0.45%	+8.0%	-1.92%	-2.0%	+0.08%	+5.1%
MPNET	-0.22%	+3.8%	+17.7%	+11.3%	-7.7%	+5.1%
Code Embedding						
Instructor	-0.22%	+7.6%	+25.7%	+31.1%	-0.14%	+1.8%
CodeBert	-41.7%	+43.7%	0.0%	0.0%	+62.5%	0.0%
CodeT5-small	-69.5%	-20.0%	-100.0%	+43.9%	-72.9%	-15.3%
Win Percentage	11.1%	88.9%	55.6%	55.6%	33.3%	66.7%

Experiments

- ❖ For non-python datasets, the performance is on-par with keyword/code embedding based retrieval.
- ❖ For text-embedding retrieval models, we found Code2JSON to perform sub-par.
- ❖ We hypothesize that this contrast is due to a lower representation of these programming languages in the training data of text-embedding models.

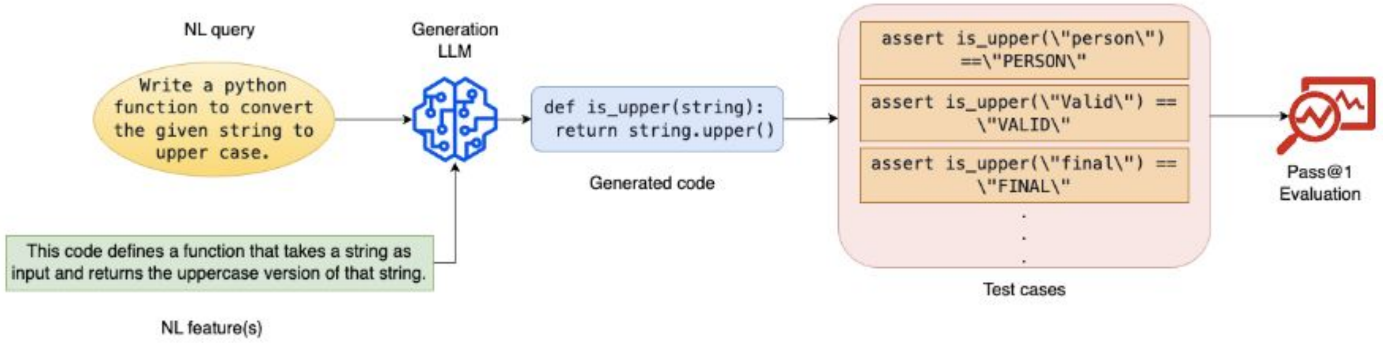
Table 2: Relative improvement in **NDCG@10** from using CODE2JSON feature extraction on non-Python datasets. Each cell shows the percentage change (positive or negative) relative to indexing raw code. The final row indicates how often the CODE2JSON-based approach outperforms direct code indexing.



	CSN(Ruby) (53K)	**HumanEval-X(cpp)** (164)	**HumanEval-X(go)** (164)	**HumanEval-X(java)** (164)	**HumanEval-X(js)** (164)
			Keyword Matching		
BM25	+213.7%	-17.3%	+14.6%	+29.9%	+7.2%
			Code Embedding		
Instructor	-6.98%	-7.59%	-10.7%	-10.9%	-6.7%
CodeBert	-75.0%	+25.7%	-7.88%	+50.6%	+4.9%
CodeT5-small	+32.4%	+9.5%	+8.9%	+2.4%	+94.6%
Win Percentage	50.0%	50.0%	50.0%	75.0%	75.0%

Experiments

❖ We also test corresponding RAG performance by using pass@1 metric for the generated code.


NL query
Write a python function to convert the given string to upper case.
Generation LLM
def is_upper(string):
  return string.upper()
Generated code
assert is_upper(\"person\") ==\"PERSON\"
assert is_upper(\"Valid\") == \"VALID\"
assert is_upper(\"final\") == \"FINAL\"
Test cases
Pass@1 Evaluation
This code defines a function that takes a string as input and returns the uppercase version of that string.
NL feature(s)

❖ The retrieved natural language descriptions act as guiding signals to the generator LLM for code generation.

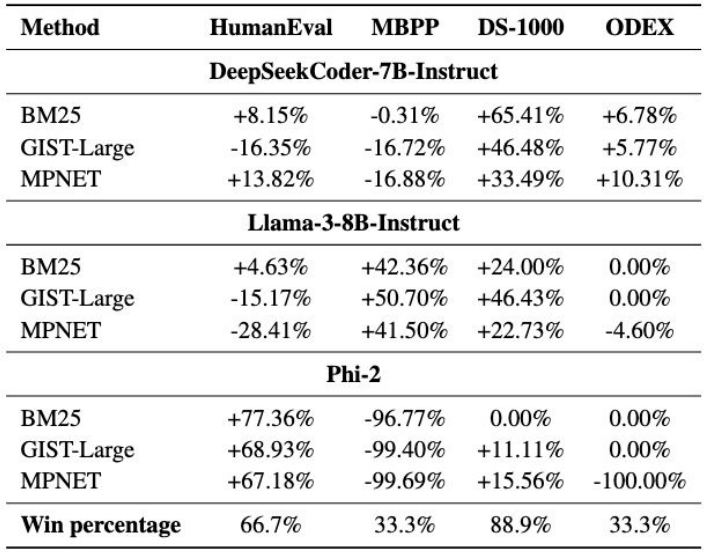
Experiments

- ❖ Comparison is done towards code generation using the retrieved natural language descriptions vs code snippets.

- ❖ Code2JSON-aided generation is superior for Llama-3-8B-Instruct and DeepSeekCoder-7B-Instruct models.

- ❖ This suggests that relatively bigger models benefit more from rich-semantic context.

Table 3: Relative improvement in code generation (**Pass@1**) scores. Each cell shows the percentage increase (or decrease) in performance when using CODE2JSON (descriptions) over the baseline (raw code). The final row indicates the proportion of instances where CODE2JSON outperforms the regular RAG workflow.



Method	**HumanEval**	**MBPP**	**DS-1000**	**ODEX**
	DeepSeekCoder-7B-Instruct			
BM25	+8.15%	-0.31%	+65.41%	+6.78%
GIST-Large	-16.35%	-16.72%	+46.48%	+5.77%
MPNET	+13.82%	-16.88%	+33.49%	+10.31%
	Llama-3-8B-Instruct			
BM25	+4.63%	+42.36%	+24.00%	0.00%
GIST-Large	-15.17%	+50.70%	+46.43%	0.00%
MPNET	-28.41%	+41.50%	+22.73%	-4.60%
	Phi-2			
BM25	+77.36%	-96.77%	0.00%	0.00%
GIST-Large	+68.93%	-99.40%	+11.11%	0.00%
MPNET	+67.18%	-99.69%	+15.56%	-100.00%
Win percentage	66.7%	33.3%	88.9%	33.3%

Cost analysis

❖ We found for simple BM25 indexing, Code2JSON yields indexes that require a lower storage capacity while performing on par or better than conventional code-indexing.

Dataset	Raw Code Index size	Feature Index size
HumanEval	248KB	176KB
MBPP	396KB	584KB
DS-1000	72MB	22MB
ODEX	72MB	22MB

❖ An approximate cost analysis is also provided for researchers and engineers to assess the feasibility of Code2JSON for further enhancements.

Dataset (size)	Time(s)	Input / Output tokens	Total Cost($)	Cost (1M tokens)($)
HumanEval (164)	86.11	35.3K / 12.8K	0.23	4.78
MBPP (964)	470.14	85.7K / 63.2K	0.93	6.24
CSN (53270)	72486.08	7.6M / 6.4M	69.66	5.70

Additional Experiments

- ❖ Low-Rank Adaptation(LORA) fine-tuning
- ❖ Ablation study with a smaller model for Code2JSON
- ❖ **Application of Code2JSON for unstructured code repositories**

(Discussed in Paper Appendix)

Limitations

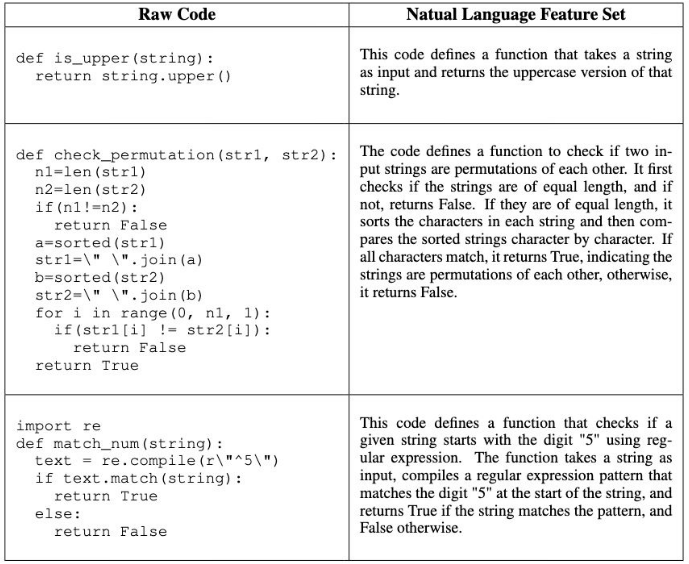
- **Context Length Constraints** - The LLM's context window poses a limitation for large code files. Chunking can lead to loss of contextual dependencies, necessitating hierarchical retrieval methods or chunk-aware generation strategies.

- **Lack of Ground Truth Annotations** - While our experiments were conducted on widely used benchmark datasets, real-world adoption requires further validation on diverse and unstructured repositories. This requires high-quality ground truth data on real-world codebases.

Conclusion

- ❖ Our experiments on ~125K code snippets across multiple programming languages demonstrate that Code2JSON can **improve or at least match retrieval and code generation quality** in many scenarios, even under resource constraints.

- ❖ Despite before-mentioned challenges, CODE2JSON demonstrates a promising step toward **structured feature extraction** for code retrieval-augmented generation (code RAG) systems.



Raw Code	Natual Language Feature Set

`def is_upper(string):`
`  return string.upper()`
	This code defines a function that takes a string as input and returns the uppercase version of that string.

`def check_permutation(str1, str2):`
`  n1=len(str1)`
`  n2=len(str2)`
`  if(n1!=n2):`
`    return False`
`  a=sorted(str1)`
`  str1=\" \".join(a)`
`  b=sorted(str2)`
`  str2=\" \".join(b)`
`  for i in range(0, n1, 1):`
`    if(str1[i] != str2[i]):`
`      return False`
`  return True`
	The code defines a function to check if two input strings are permutations of each other. It first checks if the strings are of equal length, and if not, returns False. If they are of equal length, it sorts the characters in each string and then compares the sorted strings character by character. If all characters match, it returns True, indicating the strings are permutations of each other, otherwise, it returns False.

`import re`
`def match_num(string):`
`  text = re.compile(r\"^5\")`
`  if text.match(string):`
`    return True`
`  else:`
`    return False`
	This code defines a function that checks if a given string starts with the digit "5" using regular expression. The function takes a string as input, compiles a regular expression pattern that matches the digit "5" at the start of the string, and returns True if the string matches the pattern, and False otherwise.