Deep Learning

C-SHARP STARCODER2-15B

Presented by
Shah Yash-yks,
Paliwal Durvish-dp2225,
Kadam, Pruthvi Vinayak-pk759

INTRODUCTION

This project adapts the StarCoder2 Self-Alignment Pipeline to generate high-quality instruction-response datasets specifically for the C# programming language using Stack v2 data.

To support the fine-tuning of large language models (LLMs) by automatically generating diverse and useful instructional data outside of Python, broadening model applicability across programming languages.

SEED DATASET CURATION

Parsing & Language Filtering

We begin by ingesting the raw Stack v2 corpus and isolating C#-specific entries. Using the Hugging Face Datasets library in streaming mode, we extract only those snippets whose metadata or file extension indicates C#. This early filtering ensures that all downstream operations—particularly those that rely on C# syntax—are applied only to relevant samples, saving time and computational resources.

SEED DATASET CURATION

SHA1-Based Deduplication

With potentially millions of code samples, redundancy is inevitable. To eliminate duplicates, we process the data in batches of 250 snippets: compute a SHA1 hash of each code string, and maintain a running set of seen hashes. When a new hash matches an existing one, the sample is discarded; otherwise, it’s appended to our growing collection. This approach scales linearly with data size and guarantees global uniqueness across the dataset.

SEED DATASET CURATION

Finalization

After deduplication, we use Tree-sitter to structurally filter for C# functions that include a return value, ensuring each snippet is meaningful for instruction tuning. The process is done in memory-efficient batches, with SHA1 hashes used to eliminate duplicates. Valid samples are stored with unique IDs and saved in a compact .parquet file containing only id and seed, ready for the next pipeline stage.

```
[ ] ds = datasets.load_dataset(
        "bigcode/the-stack-v2-dedup",
        "C-Sharp",
        streaming=True,
        split="train"
    )
```

```
/usr/local/lib/python3.11/dist-packages/huggingface_hub/utils/_auth.py:94: UserWarning:
The secret `HF_TOKEN` does not exist in your Colab secrets.
To authenticate with the Hugging Face Hub, create a token in your settings tab (https://huggingface.co/settings/tokens), set it as secr
You will be able to reuse this secret in all of your notebooks.
Please note that authentication is recommended but still optional to access public models or datasets.
  warnings.warn(
README.md: 100% 89.2k/89.2k [00:00<00:00, 6.56MB/s]
Resolving data files: 100% 757/757 [00:00<00:00, 9.66it/s]
```

	seed	id
0	///<summary>\n/// Returns a deep copy of the c...	1
1	/// <summary>\n/// 初始化任务集合\n/// </summary>\n//...	2
2	/// <summary>\n/// Tries to convert the string...	3
3	/// <summary>\n/// Returns the concrete implem...	4
4	/// <summary>\n/// Gets the positive modulo of...	5

SELF OSS INSTRCUT

S → C (Seed to Concept)

In this stage, each C# code snippet (the “seed”) is analyzed to generate a corresponding natural language concept that describes the code’s intended function. The goal is to abstract the code into a concise, imperative instruction, like “Sort an array in ascending order” or “Check if a number is prime.” This step bridges raw code and user intent, forming the foundation for instruction-style prompting.

```
MAX_NEW_DATA=1000000

!python /content/starcoder2-self-align/src/star_align/self_ossinstruct.py \
  --seed_data_files "/content/seed2_output_cleaned.parquet" \
  --use_vllm_server True \
  --instruct_mode "S->C" \
  --max_new_data $MAX_NEW_DATA \
  --tag concept_gen \
  --temperature 0.7 \
  --seed_code_start_index 0 \
  --model bigcode/starcoder2-15b \
  --num_fewshots 8 \
  --num_batched_requests 500 \
  --num_sample_per_request 1 \
  --async_micro_batch_size 20 \
  --delay 0
```

Example 8
Snippet
```
public static format_size(num):
    """http://stackoverflow.com/a/1094933
    """
    for x in ['bytes', 'KB', 'MB', 'GB']:
        if num < 1024.0 and num > -1024.0:
            return "%3.1f%s" % (num, x)
        num /= 1024.0
    return "%3.1f%s" % (num, 'TB')
// assert format_size(1024**2 - 1) == '1024.0KB'
// assert format_size(1024*512) == '512.0KB'
```

Concepts
arithmetic operations for size conversion, rounding numbers, dynamic unit selection, string interpolation

SELF OSS INSTRCUT

C → I (Concept to Instruction)

Once we have a concept, we convert it into a more user-friendly and complete instruction. This involves rewriting terse commands into full prompts that align with how users typically ask questions or give tasks to AI models—e.g., “Write a C# function that checks whether a number is prime.” This transformation enhances clarity and usability, making the prompt more natural and instructional.

```
MAX_NEW_DATA=1000000

!python /content/starcoder2-self-align/src/star_align/self_ossinstruct.py \
  --seed_data_files "/content/starcoder2-self-align/src/star_align/data-concept_gen-s_c-78ce2-0-20250515_045332.jsonl" \
  --use_vllm_server True \
  --instruct_mode "C->I" \
  --max_new_data $MAX_NEW_DATA \
  --tag concept_gen \
  --temperature 0.7 \
  --seed_code_start_index 0 \
  --model bigcode/starcoder2-15b \
  --num_fewshots 8 \
  --num_batched_requests 500 \
  --num_sample_per_request 1 \
  --async_micro_batch_size 20 \
  --delay 0
```

Task
Write a Python function `huffman_decompress` that takes two arguments: `compressed_data`, a byte array of 0/1 sequence representing Hu

Example 8
Properties
category: function implementation
language: C//
difficulty: easy
concepts: string manipulation and parsing, list comprehension, iterative list extension, handling text connectors

SELF OSS INSTRCUT

I → R (Instruction to Response)

In the final step, we generate a response to the instruction using an LLM (e.g., StarCoder2) served through a vLLM-compatible API. The model receives the full instruction and outputs a matching C# function. This response, when paired with the instruction, completes the instruction-response pair. These finalized pairs form the dataset used for fine-tuning instruction-following models.

```
MAX_NEW_DATA=1000000

!python /content/starcoder2-self-align/src/star_align/self_ossinstruct.py \
  --seed_data_files "/content/starcoder2-self-align/src/star_align/data-concept_gen-c_i-f4b55-0-20250515_051008.jsonl" \
  --use_vllm_server True \
  --instruct_mode "I->R" \
  --max_new_data $MAX_NEW_DATA \
  --tag concept_gen \
  --temperature 0.7 \
  --seed_code_start_index 0 \
  --model bigcode/starcoder2-15b \
  --num_fewshots 8 \
  --num_batched_requests 500 \
  --num_sample_per_request 1 \
  --async_micro_batch_size 20 \
  --delay 0
```

<tests>
To test the program, we can verify that the values in each cluster are within the expected range

```csharp
public static test_segmentation(size, break_points):
    clusters = create_clusters(size, break_points)
    for cluster in clusters:
        // assert np.all(cluster >= 0) and np.all(cluster <= 1)
    break_points = [0] + break_points + [size]
    for i in range(break_points) - 1):
        // assert clusters[i]) == break_points[i + 1] - break_points[i]

// Test cases
test_cases = [
    (20, [5, 10, 15]),
    (10, [3, 6, 8]),
    (15, [3, 6, 12]),
    (30, [7, 14, 21, 27]),
    (5, [1]),
    (10, []),
    (50, [10, 20, 30, 40]),
]

for size, breakpoints in test_cases:
    test_segmentation(size, breakpoints)
```

</tests>

SELF VALIDATION

After generating instruction-response pairs, we perform self-validation to ensure quality and consistency. The validation step checks whether the generated response correctly and fully addresses the given instruction. This can involve using automated filters such as syntax checks, output comparisons, or return-value detection, and may also include heuristics like matching key terms or verifying code structure. The goal is to filter out low-quality or incomplete responses, ensuring that only accurate and useful pairs are retained for training.

```
--- Example 3 ---
[Function Snippet]:
public static EvaluateExpression(string expression):
    double result = 0.0;
    int num1 = 0, num2 = 0;
    string op = "";

    string[] tokens = expression.Split(" ");
    foreach (string token in tokens:
        if double.TryParse(token, out double num):
            if op == "+":

[Test Snippet]:
// assert EvaluateExpression("22 + 8 * 2 / 2") == 26.0
// assert EvaluateExpression("22 + 8 * 2") == 42
// assert EvaluateExpression("22 + 8") == 30
// assert EvaluateExpression("22 + 8 * 2") == 42
// assert EvaluateExpression("22 + 8 * 2 / 2") == 26.0
```

✅ Total responses checked: 978
📌 Valid C# blocks (code + test): 942
⚠️ Missing or incomplete pairs: 36

THANK YOU