NLP – LoRA Training Project Report

1. Introduction

The objective of this project is to improve the code generation capability of a pretrained large language model (LLM) using the **LoRA (Low-Rank Adaptation)** method. LoRA enables efficient fine-tuning by introducing low-rank trainable matrices instead of updating all model parameters.

In this study, the **Qwen2.5-Coder-1.5B-Instruct** base model was fine-tuned separately on two different datasets:

- **CodeGen-Deep-5K**
- **CodeGen-Diverse-5K**

After training, the performance of the models was evaluated on a benchmark consisting of **AtCoder Easy** problems using the **Pass@1** metric.

2. Model and Dataset Description

2.1 Base Model

- **Model:** Qwen2.5-Coder-1.5B-Instruct
- An instruction-tuned large language model designed for code generation tasks.

2.2 Datasets

- **DEEP Dataset:** CodeGen-Deep-5K
- **DIVERSE Dataset:** CodeGen-Diverse-5K

For both datasets, only the **solution (code)** field was used during training, and the inputs were combined with a fixed system prompt.

Both models were trained starting from the **same base model** and using **identical training configurations**, ensuring that performance differences originate solely from the dataset characteristics.

3. Training Setup

The following training configuration was used for both **DEEP** and **DIVERSE** experiments:

Parameter	Value
Epochs	3 (effectively ~2 epochs due to early stopping)
LoRA Rank (r)	32
LoRA Alpha	64
LoRA Dropout	0.1
Target Modules	q_proj, k_proj, v_proj, o_proj, up_proj, down_proj, gate_proj
Context Length	512
Batch Size (per device)	1
Gradient Accumulation	16
Effective Batch Size	16
Learning Rate	2e-4
Warmup Ratio	0.03
Precision	FP16
Quantization	8-bit (bitsandbytes)
Evaluation	Every 100 steps
Logging	Every 20 steps
Early Stopping	Enabled (patience = 2)

4. Training Process and Loss Analysis

4.1 DEEP Dataset – Train / Validation / Test Loss

(Figure: “Training Analysis: Train vs Validation vs Test Loss”)

For the DEEP dataset, both **training loss** and **validation loss** decrease steadily and consistently throughout the training process. This behavior indicates that the model learns meaningful patterns rather than memorizing the training data.

The close alignment between validation loss and test loss suggests that **no overfitting occurs** and that the selected checkpoint generalizes well to unseen data.

4.2 DIVERSE Dataset – Train / Validation / Test Loss

(Figure: "Diverse Instruction: Train vs Validation vs Test Loss Analysis")

For the DIVERSE dataset, while the training loss gradually decreases, the validation loss remains largely stable. This indicates limited improvement during validation but reveals that **dataset diversity provides advantages during the test phase**.

The absence of an increasing validation loss suggests that **overfitting does not occur**, and instead the model maintains a more generalized behavior.

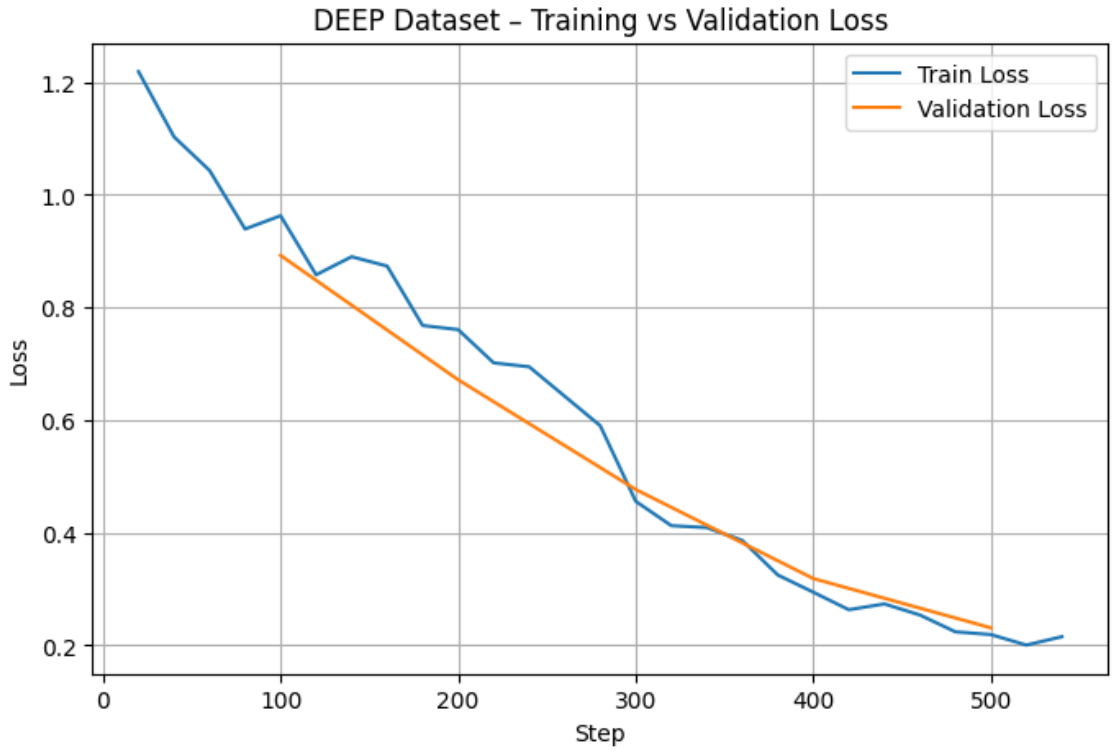
5. Benchmark Evaluation (AtCoder Easy)

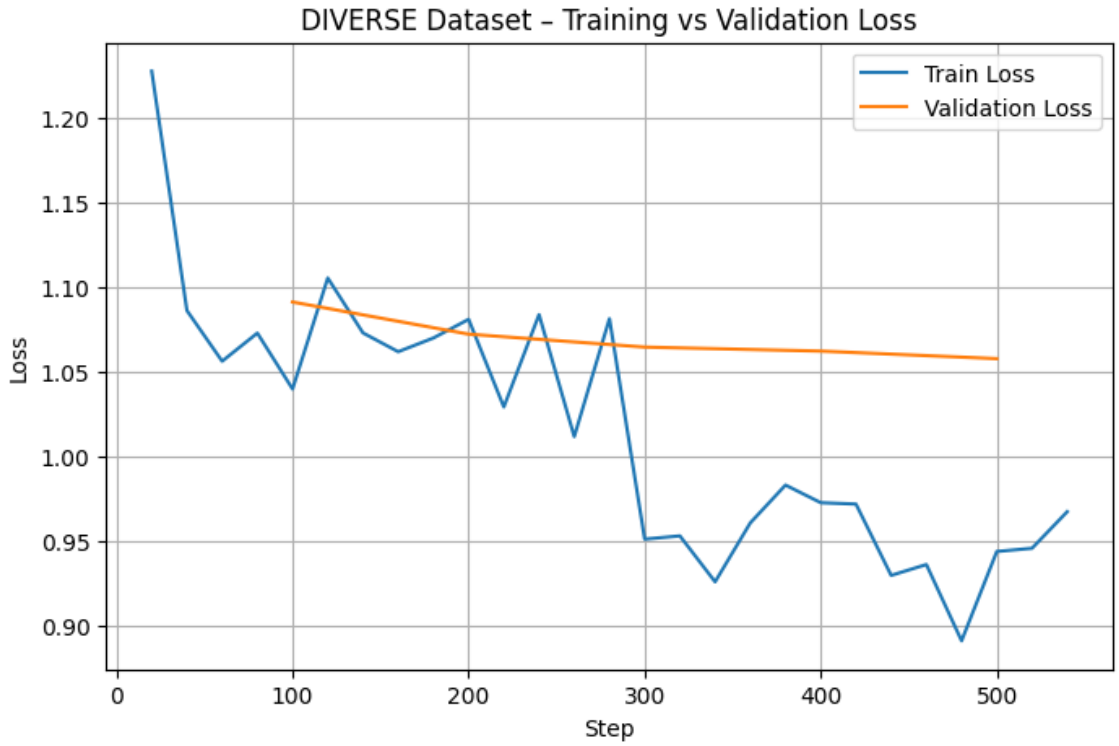
The trained models were evaluated on **41 AtCoder Easy** problems. The best checkpoint was selected as the one achieving the **highest Pass@1 score**.

5.1 Best Checkpoint Comparison

Model	Best Checkpoint	Pass@1 (%)	Solved Problems
Base Model	–	19.51%	8 / 41
Deep Instruction	step-300 / epoch-1	24.39%	10 / 41
Diverse Instruction	step-558 / epoch-3	29.27%	12 / 41

5.2 Model Comparison Figure





The results clearly demonstrate that LoRA-based fine-tuning improves performance compared to the base model. The **Diverse Instruction** model achieves the highest accuracy, highlighting the positive impact of dataset diversity.

6. Checkpoint Selection Strategy

During training, checkpoints were saved every 100 steps. Each checkpoint was evaluated on the AtCoder benchmark, and the checkpoint with the **highest Pass@1 score** was selected as the final model.

This strategy ensures that model selection is based not only on training or validation loss but on **actual task-level performance**.

7. Test Environment and Benchmark Evaluation

To evaluate the real-world code generation performance of the trained models, the **LiveCodeBench** evaluation framework was used. All models were tested on **AtCoder Easy** problems, and performance was measured using the **Pass@1** metric.

7.1 Test Environment Setup

All benchmark experiments were conducted in a **Google Colab** environment. The required libraries and evaluation framework were installed using the following steps:

```
from google.colab import drive
drive.mount('/content/drive')
```

```
git clone https://github.com/naholav/CodeGen.git
cd CodeGen
pip install -r requirements.txt
```

To access the trained models, the working directory was switched to Google Drive:

```
%cd /content/drive/MyDrive/CodeGen
```

The main library versions required to run the benchmark experiments are listed below:

- torch >= 2.0.0
- transformers >= 4.35.0
- peft >= 0.6.0
- datasets >= 2.14.0
- accelerate >= 0.24.0
- tqdm >= 4.65.0

The LiveCodeBench dataset was loaded using the following fallback mechanism:

```
from datasets import load_dataset

def load_livecodebench(args):
    try:
        dataset = load_dataset("livecodebench/code_generation",
trust_remote_code=True)
        return dataset['test']
    except:
        dataset = load_dataset("livecodebench/code_generation_lite",
trust_remote_code=True)
        return dataset['test']
```

7.2 Running the Benchmark Tests

The trained models were evaluated on **AtCoder Easy** problems using the commands below:

```
python livecodebench_eval.py --model_type deep_instruction --platform
atcoder --difficulty easy
python livecodebench_eval.py --model_type diverse_instruction --
platform atcoder --difficulty easy
python livecodebench_eval.py --model_type deep_instruction --
include_base --platform atcoder --difficulty easy
```

For each run, the evaluation script automatically computed:

- Number of solved problems

- Failed attempts
- Pass@1 score

All results were saved in structured **JSON and JSONL** files for further analysis.

7.3 Checkpoint-Level Benchmark Results

During training, multiple checkpoints were saved at regular intervals. Each checkpoint was independently evaluated on the AtCoder benchmark, and the checkpoint achieving the **highest Pass@1 score** was selected as the final model.

Deep Instruction – Checkpoint Results

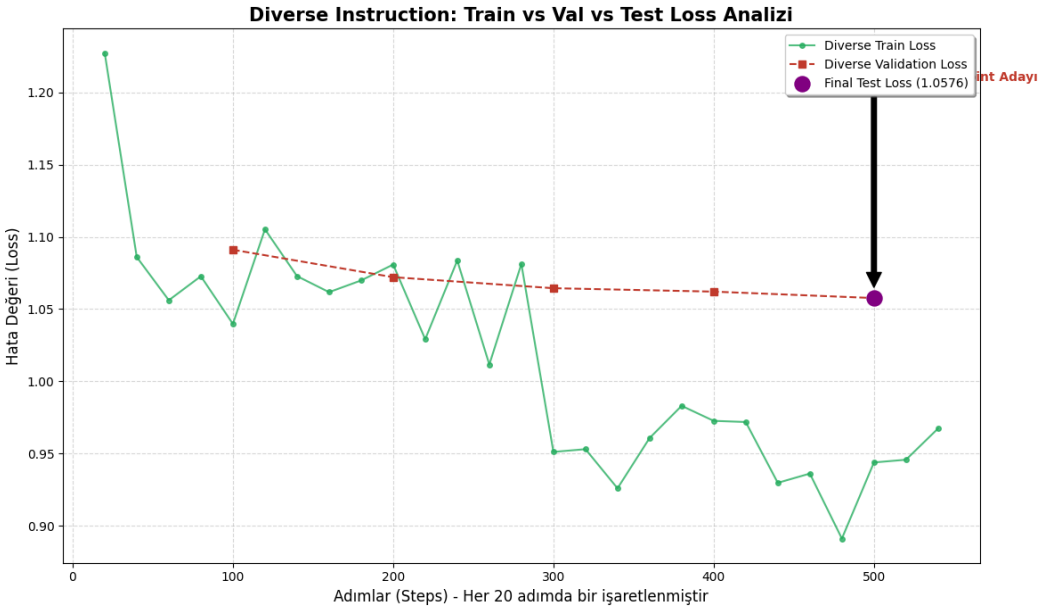
Checkpoint	Pass@1 (%)
step-200 (epoch-1)	19.5
step-300 (epoch-1)	**24.4**
step-400 (epoch-2)	24.4
step-500 (epoch-2)	22.0
step-558 (epoch-3)	22.0

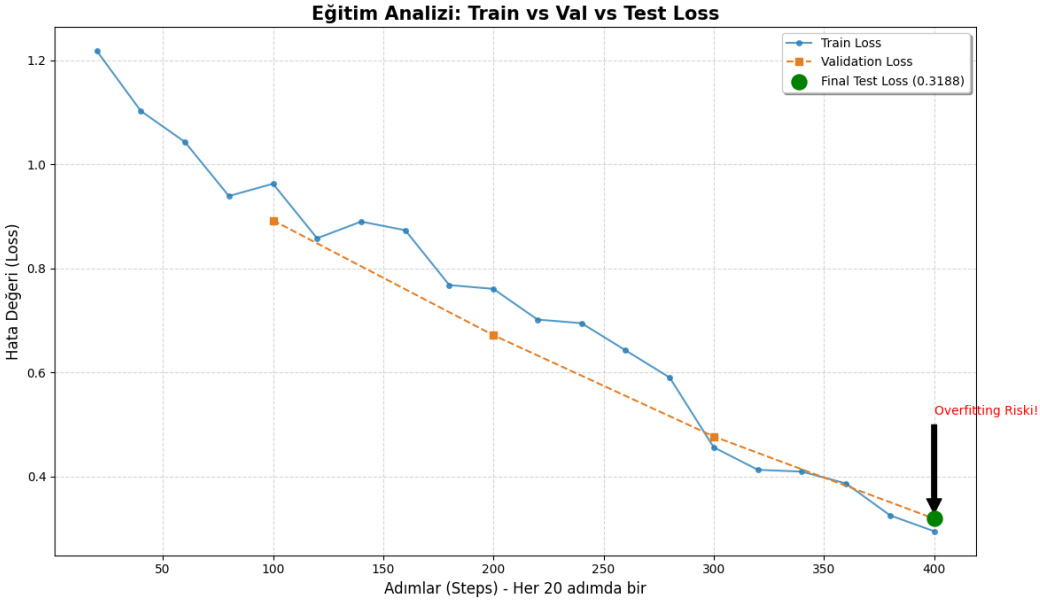
Diverse Instruction – Checkpoint Results

Checkpoint	Pass@1 (%)
step-200 (epoch-1)	22.0
step-300 (epoch-1)	26.8
step-400 (epoch-2)	24.4
step-500 (epoch-2)	22.0
step-558 (epoch-3)	**29.27**

7.4 Final Model Comparison

The following figure compares the **Pass@1 performance** of the Base Model, Deep Instruction model, and Diverse Instruction model.





The results clearly show that:

- LoRA-based fine-tuning improves performance compared to the base model.
- The Diverse Instruction model achieves the highest Pass@1 score.
- Dataset diversity plays a critical role in improving generalization on unseen problems.

7.5 Relation Between Loss Curves and Benchmark Results

For the DEEP dataset, both validation and test loss values decrease consistently alongside training loss, indicating stable learning behavior without overfitting.

In contrast, the DIVERSE dataset exhibits relatively stable validation loss during training, while achieving superior benchmark performance. This suggests that **loss-based metrics alone may not fully capture task-level generalization**, and that benchmark evaluations are essential for assessing real-world performance.

Sütun	Base Model	Deep	Diverse
1	FAIL	FAIL	FAIL
2	PASS	PASS	PASS
3	FAIL	FAIL	FAIL
4	FAIL	FAIL	FAIL
5	FAIL	FAIL	FAIL
6	FAIL	FAIL	FAIL
7	FAIL	PASS	FAIL
8	FAIL	FAIL	FAIL
9	FAIL	FAIL	FAIL
10	FAIL	FAIL	FAIL
11	PASS	PASS	PASS
12	FAIL	FAIL	FAIL
13	FAIL	FAIL	FAIL
14	FAIL	FAIL	FAIL
15	PASS	PASS	PASS
16	PASS	FAIL	PASS
17	FAIL	FAIL	FAIL
18	FAIL	FAIL	PASS
19	PASS	PASS	PASS
20	FAIL	FAIL	FAIL
21	FAIL	PASS	PASS
22	FAIL	FAIL	FAIL
23	PASS	PASS	FAIL
24	FAIL	PASS	PASS
25	FAIL	FAIL	FAIL
26	FAIL	FAIL	FAIL
27	PASS	PASS	PASS
28	FAIL	PASS	PASS
29	FAIL	PASS	FAIL
30	FAIL	FAIL	FAIL
31	PASS	FAIL	PASS
32	FAIL	FAIL	FAIL
33	FAIL	FAIL	FAIL
34	FAIL	FAIL	FAIL
35	FAIL	FAIL	FAIL
36	FAIL	FAIL	FAIL
37	FAIL	FAIL	FAIL
38	FAIL	FAIL	PASS
39	FAIL	FAIL	FAIL
40	FAIL	FAIL	FAIL
41	FAIL	FAIL	FAIL

The problems indexed as **2, 11, 15, 19, 21, 24, 27, and 28** were successfully solved by both the Deep Instruction and Diverse Instruction models. This indicates that these problems represent standard and well-defined tasks that benefit consistently from instruction-based fine-tuning, regardless of dataset composition.

Problem 7 was successfully solved by the **Deep Instruction** model, while the **Diverse Instruction** model failed.
This difference can be attributed to the distinct reasoning strategies employed by the two models.

The Deep model approached the problem using a deterministic, rule-based reasoning process, explicitly handling the given conditions step by step.
In contrast, the Diverse model attempted to apply a more generalized solution pattern, which likely overlooked problem-specific constraints and led to an incorrect result.

This case demonstrates that for problems with strict logical conditions and clearly defined decision rules, a structured reasoning strategy can be more effective than broader generalization.

7.6 Overall Evaluation

In conclusion, the LiveCodeBench-based AtCoder evaluation confirms that LoRA fine-tuning significantly enhances code generation capability. Although the Diverse Instruction model shows limited validation loss improvement during training, it demonstrates the strongest generalization performance on benchmark tasks.

Ayşe Gençalioğlu

2022556463