Project Part 2

Generative AI

Saarland University – Winter Semester 2024/25

Gaygusuz Osman
7065438
osga00001@stud.uni-saarland.de

1 Exercise Implementation: I.11

The results obtained for generating repairs using the `Phi-3-SFT-Repair-r16-alpha32` and `Phi-3-SFT-Hints-r16-alpha32` models respectively as Repair and Hints on the table:

Problem	**RPass (%)**		**REdit**		**Overall Metrics**			
	Repair	Hints	Repair	Hints	Overall RPass (%)		Overall REdit	
					Repair	Hints	Repair	Hints
Problem 1	80.0	60.0	16.50	29.00				
Problem 2	60.0	80.0	04.00	37.75				
Problem 3	80.0	80.0	15.25	15.00	68.0	72.0	10.41	19.44
Problem 4	60.0	60.0	04.33	06.66				
Problem 5	60.0	80.0	08.33	08.00				

Preprint. Under review.

2 Exercise Implementation: I.12

The results obtained for generating hints using the `Phi-3-SFT-Repair-r16-alpha32` and `Phi-3-SFT-Hints-r16-alpha32` models respectively as Repair and Hints on the table:

Table 1: Hint Generation Results Per Problem for Two Models

Problem	**HCorrect (%)**		**HInformative (%)**		**HConceal (%)**		**HComprehensible (%)**		**HGood (%)**	
	Repair	Hints	Repair	Hints	Repair	Hints	Repair	Hints	Repair	Hints
Problem 1	60.0	80.0	60.0	60.0	80.0	100.0	60.0	60.0	40.0	40.0
Problem 2	80.0	100.0	80.0	100.0	100.0	100.0	60.0	100.0	60.0	100.0
Problem 3	100.0	80.0	100.0	80.0	60.0	100.0	100.0	80.0	60.0	80.0
Problem 4	40.0	60.0	40.0	60.0	100.0	100.0	40.0	60.0	40.0	60.0
Problem 5	80.0	80.0	80.0	80.0	100.0	100.0	60.0	80.0	60.0	80.0

Table 2: Overall Hint Generation Results for Two Models

Metric	**Repair (%)**	**Hints (%)**
HCorrect	72.0	80.0
HInformative	72.0	76.0
HConceal	88.0	100.0
HComprehensible	64.0	76.0
HGood	52.0	72

3 Exercise Implementation: I.13

For this question we evaluate the three different fine-tuned models with varying (r, alpha) on program repair. Here are the results for the program repair with the different parameter configurations:

Table 3: Comparative Repair Results Per Problem for Three Configurations of Phi-3-mini model

Problem	**RPass (%)**			**REdit**		
	(4,8)	(16,32)	(64,128)	(4,8)	(16,32)	(64,128)
Problem 1	60.0	80.0	100.0	17.00	16.50	16.80
Problem 2	60.0	100.0	80.0	4.00	29.40	27.75
Problem 3	60.0	80.0	80.0	16.00	12.75	13.00
Problem 4	60.0	60.0	80.0	6.33	5.33	7.00
Problem 5	60.0	100.0	100.0	10.66	7.60	5.60

Table 4: Overall Repair Metrics for Three Models

Metric	**Phi-3-mini (4,8)**	**Phi-3-mini (16,32)**	**Phi-3-mini (64,128)**
Overall RPass (%)	60.0	84.0	88.0
Overall REdit	10.80	15.14	13.77

Table 5: Resource Usage for LoRA Configurations

Metric	$(4, 8)$	$(16, 32)$	$(64, 128)$
Number of Trainable Parameters	$7,471,104$	$29,884,416$	$119,537,664$
Training Time (minutes)	40.52	39.99	44.03
Training Memory (GB)	0.846	0.983	1.625

The results demonstrate a significant improvement in model performance after fine-tuning with different configurations of (r, α) compared to the base Phi-3-mini model in project part 1.

- **RPass Improvement:** The `RPass` values for all configurations are nearly double compared to the base `Phi-3-mini` model. For instance:
  - The overall `RPass` metric increased from 36.0% to 88.0% $((64, 128))$.
- **REdit Reduction:** Similarly, the `REdit` values decreased significantly across all problems, indicating better-quality repairs. For example:
  - The overall `REdit` dropped from 18.1 (base model) to 13.77 $((64, 128))$.

Fine-tuning required minimal computational resources:

- Training time was less than 45 minutes for all configurations.
- Memory usage remained below 2 GB, with $(4, 8)$ configuration requiring only 0.846 GB.

Despite these low computational costs, fine-tuning delivered substantial performance gains.

The results highlight the power of fine-tuning, even with limited resources. Improvements in `RPass` and `REdit` illustrate that the model becomes more effective at generating high-quality repairs while remaining computationally efficient.

4 Exercise Implementation: I.14

In this question, we analyze the tradeoffs between the LoRA parameter configurations (r, α), focusing on program repair quality metrics (`RPass`, `REdit`) and resource requirements (`Training Memory`, number of trainable parameters). The configurations analyzed are $(4, 8)$, $(16, 32)$, and $(64, 128)$.

First, let's analyze the impact of the different configurations on the quality metrics (See Table 4):

- $(4, 8)$ **Configuration:** This configuration achieves an `RPass` of 60.0%, the lowest among all configurations, and the smallest overall `REdit` (10.80).
- $(16, 32)$ **Configuration:** With a significant jump in `RPass` (84.0%), this configuration balances repair quality and resource usage. It shows a moderate increase in `REdit` (15.14).
- $(64, 128)$ **Configuration:** This configuration achieves the best `RPass` (88.0%) and a slightly reduced `REdit` (13.77) compared to $(16, 32)$.

The resource requirements for each configuration are shown in Table 5:

- $(4, 8)$**:** This configuration uses the least memory $(0.846$ GB) and requires the fewest trainable parameters $(7,471,104)$. It is the most computationally efficient, making it suitable for resource-constrained environments.
- $(16, 32)$**:** While using more memory $(0.983$ GB) and parameters $(29,884,416)$, it maintains a comparable training time to $(4, 8)$ and achieves a significant improvement in repair quality. By adding only 0.14 GB to the memory usage, we can multiply the number of parameters by 4.
- $(64, 128)$**:** The most resource-intensive configuration, requiring 1.625 GB of memory and $119,537,664$ parameters. However, it provides the best repair quality. Compared to $(4, 8)$, we double the training memory but increase the number of parameters by approximately 16.

Tradeoffs and Observations

- The $(4, 8)$ configuration is highly resource-efficient but has limited improvements in `RPass`.
- The $(16, 32)$ configuration strikes a balance between quality and resource usage, making it a versatile choice for most scenarios. Compared to $(4, 8)$, it is also very resource-efficient but performs better.
- The $(64, 128)$ configuration maximizes repair quality but at a significant cost in terms of memory and trainable parameters. It is best suited for scenarios where high-quality repairs are critical, and resource constraints are not a primary concern.

Conclusion

While $(4, 8)$ is the most efficient, $(64, 128)$ provides the best results at a higher resource cost. The $(16, 32)$ configuration represents a middle ground, achieving substantial improvements in quality without significantly increasing computational demands.

5 Exercise Implementation: I.15

Here the program repair results for the model trained on the unified dataset:

Problem	RPass (%)	REdit
Problem 1	80.0	19.00
Problem 2	80.0	31.75
Problem 3	100.0	14.20
Problem 4	60.0	5.00
Problem 5	80.0	11.00

Table 6: Repair Generation Results Per Problem Using `phi-3-mini` trained on unified dataset (16,32)

Metric	Value
Overall Correct Rate (%)	80.0
Overall Avg. Edit Distance	16.65

Table 7: Overall Repair Generation Results Using `phi-3-mini` trained on unified dataset (16,32)

6 Exercise Implementation: I.16

Here the evaluation of the multitask model on hint generation:

Problem	HCorrect (%)	HInformative (%)	HConceal (%)	HComprehensible (%)	HGood (%)
Problem 1	100.0	80.0	100.0	100.0	80.0
Problem 2	80.0	80.0	100.0	80.0	80.0
Problem 3	100.0	100.0	100.0	100.0	100.0
Problem 4	80.0	80.0	100.0	80.0	80.0
Problem 5	100.0	100.0	100.0	100.0	100.0

Table 8: Hint Generation Results Per Problem Using `Phi-3-mini` trained on unified dataset (16,32)

Metric	Value (%)
HCorrect	92.0
HInformative	88.0
HConceal	100.0
HComprehensible	92.0
HGood	88.0

Table 9: Hint Generation Results Per Problem Using `Phi-3-mini` trained on unified dataset (16,32)

7 Exercise Implementation: I.17

The multitask model was designed to handle both:

- **Repair Generation:** Correct buggy code.
- **Hint Generation:** Provide meaningful hints for buggy code.

We were asked to create a combined dataset by combining training data for both tasks. The data was balanced to prevent any dominance of one task over the other. Additionally, the dataset was shuffled to ensure randomness, improving training generalization.

The following table compares the training time and memory usage for the specialized repair model and the multitask model:

Metric	Specialized Model (16,32)	Multitask Model (16,32)
Training Time (minutes)	39.99	84.02
Training Memory (GB)	0.983	0.983

Table 10: Comparison of TrainingTime and TrainingMemory

The multitask model required significantly more training time due to the larger combined dataset. However, the memory usage remained identical for the same LoRA configuration. I don't have the training memory and time for the hint specialized model but it should be relatively the same as the repair one. Indeed, the training time is doubled, however it doesn't require more training memory and is capable of performing both tasks decently.

The multitask model's performance was evaluated on both tasks, here is the comparison to the specialized models respectively for the hints and repair tasks.

Task	Metric	Specialized Model (16,32)	Multitask Model (16,32)
Hint Generation	HCorrect (%)	80.0	92.0
	HInformative (%)	76.0	88.0
	HConceal (%)	100.0	100.0
	HGood (%)	72.0	88.0
Repair Generation	Overall Correct Rate (%)	84.0	80.0
	Avg Edit Distance	15.14	16.65

Table 11: Performance Comparison for Hint and Repair Generation

Hint Generation: The multitask model outperformed the specialized hint model, achieving higher scores across all metrics.

Repair Generation: The multitask model's performance was slightly lower (Correct Rate decreased by 4%), but the difference is small, thus the model remained competitive.

Challenges: One of the challenges is to ensure a balance between repair and hint examples in the unified dataset to prevent task imbalance. Additionally, the training time increased significantly due to the dataset size.

Observations:

- The multitask model achieved superior performance in hint generation, it is most likely due to shared learning between the two tasks.
- The slight decline in repair performance may be produced by the fact that the dataset is more diverse but the diversity doesn't really help to generate better repair programs.

The multitask model has significantly improved in hint generation while having good performance in repair generation. Although training time increased, the multitask approach offers a solution for handling both tasks efficiently.

Acknowledgements

Same as part 1