Project Part 2

Generative AI

Saarland University – Winter Semester 2024/25

Alexandru Andriţa

7069106

alan00004@stud.uni-saarland.de

1 Implementation Assignment: I11

Results reported using the repairs generated by the model **Phi-3-SFT-Repair-r16-alpha32**

Problem 1	Correct	Distance
Program 1	True	4
Program 2	True	38
Program 3	True	5
Program 4	True	19
Program 5	False	−1

Problem 3	Correct	Distance
Program 1	True	3
Program 2	False	−1
Program 3	True	11
Program 4	True	39
Program 5	True	4

Problem 2	Correct	Distance
Program 1	False	1
Program 2	True	6
Program 3	False	−1
Program 4	True	114
Program 5	True	5

Problem 4	Correct	Distance
Program 1	True	−1
Program 2	True	6
Program 3	True	5
Program 4	False	−1
Program 5	True	2

Problem 5	Correct	Distance
Program 1	True	1
Program 2	True	9
Program 3	True	12
Program 4	True	2
Program 5	True	12

Averages	RPass (%)	REdit
Problem 1	80	16.5
Problem 2	80	31.5
Problem 3	80	14.25
Problem 4	60	4.33
Problem 5	100	7.2
Overall	80	14.9

Results reported using the repairs generated by the model **Phi-3-SFT-Hints-r16-alpha32**

Preprint. Under review.

Problem 1	Correct	Distance
Program 1	True	23
Program 2	True	49
Program 3	True	15
Program 4	False	−1
Program 5	True	20

Problem 3	Correct	Distance
Program 1	True	7
Program 2	False	−1
Program 3	True	11
Program 4	True	36
Program 5	True	10

Problem 2	Correct	Distance
Program 1	True	1
Program 2	True	52
Program 3	True	21
Program 4	True	75
Program 5	True	−1

Problem 4	Correct	Distance
Program 1	False	−1
Program 2	True	4
Program 3	True	5
Program 4	False	−1
Program 5	True	2

Problem 5	Correct	Distance
Program 1	False	−1
Program 2	False	−1
Program 3	True	10
Program 4	True	2
Program 5	True	23

Averages	RPass (%)	REdit
Problem 1	80	26.75
Problem 2	80	37.25
Problem 3	80	16
Problem 4	60	3.66
Problem 5	60	11.66
Overall	72	20.33

Looking at the results per each problem and the overall results, the model fine-tuned on repairs has a better performance. This is expected since a repair could influence a model more than just a simple hint, which might also be really far from the ground truth (i.e. real problem(s) in the code).

2 Implementation Assignment: I12 ([1])

2.1 Analyzed metrics using Phi-3-SFT-Repair-r16-alpha32

Problem 1	HCorrect	HInformative	HConceal	HComprehensible	HGood
Program 1 hint	0	0	0	0	0
Program 2 hint	0	1	0	0	0
Program 3 hint	1	1	0	1	0
Program 4 hint	1	1	1	1	1
Program 5 hint	0	0	0	0	0

The hint for *Program 1* says *The return statement inside the loop should be outside the loop to ensure it executes after the loop finishes, otherwise it returns immediately after the first condition is met*. This is incorrect as the return in the loop is completely fine. The issue lies in the second return as the returned value is not correct. HConceal could have been considered to be 1 even though the information was not correct, but in this case, the hints gives straight away a possible solution to the issue.
The hint for *Program 2* is correct, as it specifies that there are problems while accessing the last element from the list because the loop from the buggy problem is iterating too much. I will consider this as HInformative = 1. The other fields are 0 because in the end, one proper bug from the code

is not really solved with the hint. Instead, the hint only gives a slight direction for the learner to investigate.
In case of *Program 3*, the hint is correct and informative. The downside is that it gives the solution straight away i.e. *the return statement inside the else block should be outside the for loop to ensure the loop completes before returning the final position*. Because of this, HConceal = 0.
In case of *Program 5*, the hint is incorrect and it contains only redundant information. It talks about a bug that actually does not exist. We could have considered HConceal = 1, only because the learner would reason whether this information is truly helpful or not, but it already gives a solution, even though the solution does not solve any of the bugs.

In this case, for **Problem 1**, the percentage is quite low. HGood = 20%. As a general observation, the hints were quite bad overall both in terms of hiding the solution and also in identifying the issue(s).

Problem 2	HCorrect	HInformative	HConceal	HComprehensible	HGood
Program 1 hint	1	1	0	1	0
Program 2 hint	1	1	1	1	1
Program 3 hint	1	1	1	1	1
Program 4 hint	1	1	1	1	1
Program 5 hint	1	1	0	1	0

The hint for *Program 1* is correct but it gives too much information: *In unique_day, the variable 'day' is not defined within the function; it should be replaced with the parameter 'date'*. Thus, we consider HConceal = 0.
In case of *Program 3*, again, the hint is correct. The hint is *In the unique_month function, you forgot to initialize the variable 'count' before using it to increment*. The problem is that it already tells the learner that in a specific function, a variable is not initialized. There are 2 possible ways to analyze this case: either we consider HConceal = 0 because too much information is revealed or we consider HConceal = 1 because in the end, the hint is very similar to what a Python Intrepretor would return for this kind of error. Taking into account the possible message displayed by a Python Interpretor, I agree with HConceal = 1.
The situation of *Program 5* is the same as other previous cases i.e. too much information revealed. The hint says *The issue lies in the contains_unique_day function where the return statement is incorrectly placed inside the loop, causing the function to return after checking the first day* and basically shows the solution directly. Thus, HConceal = 0.

In case of **Problem 2**, the percentage is much better i.e. HGood = 60%.

Problem 3	HCorrect	HInformative	HConceal	HComprehensible	HGood
Program 1 hint	1	1	0	1	0
Program 2 hint	0	0	0	0	0
Program 3 hint	1	1	0	1	1
Program 4 hint	1	1	1	1	1
Program 5 hint	1	1	0	1	0

The hint for *Program 2* says that the problem lies in not using `lst` when appending to the `new` list. The hint is wrong and it contains redundant information.
The hint for *Program 3* is quite direct telling the learner the call of `sort` method could affect the relative order. Thus, HConceal = 0.
The hint for *Program 5* says *You are appending an integer to a list instead of appending the element itself; use 'new_lst.append(i)' instead of 'new_lst = new_lst + i'*. Indeed, the bug and the solution are correctly described in the hint, but since the solution is presented straight away, setting HConceal = 0.

In case of **Problem 3**, the percentage is HGood = 40%. However, even though the score is not high, the hints are mainly correct. The only issue is with showing too much information.

Problem 4	HCorrect	HInformative	HConceal	HComprehensible	HGood
Program 1 hint	0	0	0	0	0
Program 2 hint	1	1	0	1	0
Program 3 hint	1	1	1	1	1
Program 4 hint	0	0	0	0	0
Program 5 hint	0	0	0	0	0

The hint for *Program 1* says that we should check for the length of the list instead of taking the first element. This is completely wrong and it does not have anything to do with the real bugs.
For *Program 2*, again, the solution (which is in fact, correct) is served straight away i.e. *The sort_age function incorrectly uses sort() which sorts the list in place and returns None, instead of returning a sorted list. Use sorted() to return a new sorted list*.
The hint for *Program 3* is *The variable 'a' is not defined; it seems to be a typo and should be 'lst' to correctly remove the largest element from the list*. I will consider the hint correct because the first half of it i.e. the comment regarding the typo and variable not defined is correct and solves a bug.
The hint for *Program 4* says that the case when the two lists do not have the same length is not covered. This is not true, the code covers this case and this is not the actual bug of the problem.
As in the previous program, for *Program 5*, the hint is referring to a bug that actually is not a real bug in our code.

In case of **Problem 4**, HGood = 20%. For this problem, most of the hints were referencing problems that never existed in the code.

Problem 5	HCorrect	HInformative	HConceal	HComprehensible	HGood
Program 1 hint	0	0	0	0	0
Program 2 hint	1	1	0	1	0
Program 3 hint	1	1	1	1	1
Program 4 hint	0	0	0	0	0
Program 5 hint	1	1	1	1	1

Hint for *Program 1* is again referring to an issue that actually does not exist.
Hint for *Program 2* is correct and informative, but it contains again too much information that basically serves the solution.
Hint for *Program 3* solves the issue for one bug. It says that the variable `new` is not initialized and it needs to. This does not help that much but in the end, the information is not wrong either. Thus, considering the attributes equal to 1.
For *Program 5*, again, the hint is not really on point, but at least it gives a starting point to the learner.

All in all, for *Problem 5*, it looks like HGood = 40%. Looks like a combination of cases i.e. hints not related at all to the bugs, hints revealing too much information and correct hints that could be used in solving the code issues.

Overall for this model, it looks like HGood = 36%.

2.2 Analyzed metrics using Phi-3-SFT-Hints-r16-alpha32

Problem 1	HCorrect	HInformative	HConceal	HComprehensible	HGood
Program 1 hint	1	1	1	1	1
Program 2 hint	1	1	1	1	1
Program 3 hint	1	1	1	1	1
Program 4 hint	1	1	1	1	1
Program 5 hint	0	0	0	1	0

The hint for *Program 5* says we need to consider the case when our value we want to insert is equal to the last element. Not only that, but also we need to take care of the case when our value is greater than all other elements. The hint is not correct in this case since it would lead the learner in the code in an area where there is no bug. We could consider HComprehensible = 1 because the hint is easy to understand, even though it does not really help with the bugs.

For **Problem 1**, HGood = 80%.

Problem 2	HCorrect	HInformative	HConceal	HComprehensible	HGood
Program 1 hint	1	1	1	1	1
Program 2 hint	0	0	0	1	0
Program 3 hint	1	1	1	1	1
Program 4 hint	1	1	1	1	1
Program 5 hint	0	0	0	1	0

Hint for *Program 2* i.e. *For the 'unique_day' function, think about how you're comparing the day in the birthday tuple to the input date. Are you comparing the right elements?* is not related to the actual bug - if counter = 0, the code is not returning False.

Lower percentage in this case comparing with *Problem 1*, but this is due to the fact that 2 hints are related to bugs that are not really bugs in the code. For **Problem 2**, HGood = 60%.

Problem 3	HCorrect	HInformative	HConceal	HComprehensible	HGood
Program 1 hint	1	1	1	1	1
Program 2 hint	1	1	1	1	1
Program 3 hint	1	1	1	1	1
Program 4 hint	1	1	1	1	1
Program 5 hint	1	1	1	1	1

For **Problem 3**, HGood = 100%. In this case, all hints were on point i.e. correct, informative and without revealing too much information.

Problem 4	HCorrect	HInformative	HConceal	HComprehensible	HGood
Program 1 hint	0	0	0	1	0
Program 2 hint	1	1	0	1	0
Program 3 hint	1	1	1	1	1
Program 4 hint	1	1	1	1	1
Program 5 hint	0	0	0	1	0

For *Program 5*, there is no issue with comparing the ages, like the hint is suggesting. The issue is represented by the places where `append` and `remove` are called.

Percentage for **Problem 4** is HGood = 40%.

Problem 5	HCorrect	HInformative	HConceal	HComprehensible	HGood
Program 1 hint	0	0	0	1	0
Program 2 hint	1	1	1	1	1
Program 3 hint	1	1	1	1	1
Program 4 hint	1	1	1	1	1
Program 5 hint	1	1	1	1	1

The hint for *Program 1* is saying there is a problem with the removal from the list. Actually, this is working without any issue and the actual bug (considering more than `k` values) is not identified.

Percentage for **Problem 5** is HGood = 80%.

Overall, for this model, the percentage is HGood = 72%. This is much higher than before (actually it is doubled) but this is expected since the model was fine-tuned using the generated feedback.

3 Implementation Assignment: I13

Important remark: In case of $(r, \alpha) = (16, 32)$, I ran the program again, so the values from I11 might be different.

Configuration	Training Time	Training memory
$(r, \alpha) = (4, 8)$	48.1 minutes	0.865 GB
$(r, \alpha) = (16, 32)$	47.3 minutes	0.0 GB
$(r, \alpha) = (64, 128)$	46.33 minutes	1.625 GB

3.1 $(r, \alpha) = (4, 8)$

Problem 1	Correct	Distance
Program 1	False	-1
Program 2	True	38
Program 3	True	5
Program 4	True	8
Program 5	False	-1

Problem 2	Correct	Distance
Program 1	True	1
Program 2	True	14
Program 3	False	-1
Program 4	False	-1
Program 5	True	5

Problem 3	Correct	Distance
Program 1	True	5
Program 2	False	-1
Program 3	False	-1
Program 4	True	39
Program 5	True	4

Problem 4	Correct	Distance
Program 1	False	-1
Program 2	True	6
Program 3	True	5
Program 4	False	-1
Program 5	False	-1

Problem 5	Correct	Distance
Program 1	True	1
Program 2	False	-1
Program 3	True	10
Program 4	False	-1
Program 5	True	23

Averages	RPass (%)	REdit
Problem 1	60	17
Problem 2	60	6.66
Problem 3	60	16
Problem 4	40	5.5
Problem 5	60	11.33
Overall	56	11.7142857

3.2 $(r, \alpha) = (16, 32)$

Problem 1	Correct	Distance
Program 1	True	4
Program 2	True	38
Program 3	True	5
Program 4	True	20
Program 5	False	-1

Problem 3	Correct	Distance
Program 1	True	1
Program 2	False	-1
Program 3	True	15
Program 4	True	39
Program 5	True	4

Problem 2	Correct	Distance
Program 1	True	1
Program 2	True	8
Program 3	False	-1
Program 4	True	114
Program 5	True	5

Problem 4	Correct	Distance
Program 1	False	-1
Program 2	True	6
Program 3	True	5
Program 4	False	-1
Program 5	True	2

Problem 5	Correct	Distance
Program 1	True	1
Program 2	True	9
Program 3	True	12
Program 4	False	-1
Program 5	True	12

Averages	RPass (%)	REdit
Problem 1	80	16.75
Problem 2	80	25.6
Problem 3	80	14.75
Problem 4	60	4.33
Problem 5	80	8.5
Overall	76	15.8421053

3.3 $(r, \alpha) = (64, 128)$

Problem 1	Correct	Distance
Program 1	True	4
Program 2	True	38
Program 3	True	5
Program 4	True	19
Program 5	True	18

Problem 3	Correct	Distance
Program 1	True	11
Program 2	False	-1
Program 3	True	9
Program 4	True	30
Program 5	True	4

Problem 2	Correct	Distance
Program 1	True	1
Program 2	True	6
Program 3	False	-1
Program 4	True	99
Program 5	True	5

Problem 4	Correct	Distance
Program 1	True	6
Program 2	True	6
Program 3	True	5
Program 4	False	-1
Program 5	True	5

Problem 5	Correct	Distance
Program 1	True	1
Program 2	True	9
Program 3	True	14
Program 4	True	2
Program 5	True	23

Averages	RPass (%)	REdit
Problem 1	100	16.8
Problem 2	80	27.75
Problem 3	80	13.5
Problem 4	80	5.5
Problem 5	100	9.8
Overall	88	14.5454545

Looking at the results from I3 from the first part of the project, I extracted the following values:

Averages	RPass (%)	REdit
Problem 1	40	14
Problem 2	0	0
Problem 3	20	10
Problem 4	80	22.5
Problem 5	40	17.5
Overall	36	18.11111111111111

We can see an immediate improvement directly with the smallest possible values of r and α. When $r = 4$ and $\alpha = 8$, REdit and RPass already have greater values i.e. RPass is greater, this means that the number of test cases passed has been improved and REdit is lower, this means that average distance between the buggy program and the solution (repaired program) is smaller, so the models are performing better. Analyzing the averages individually, we could argue that for each problem, the performance has relatively improved in the same manner. In my opinion, in this case, the model might struggle a little bit with underfitting, though, comparing it with the baseline models, as expected, the performance is better, so the underfitting is not seen here.
Moving on to the middle value of r i.e. $(r, \alpha) = (16, 32)$, even though the distance has increased slightly, we could see that the model has better generalization power as RPass has increased significantly. REdit is quite close to the REdit of the baseline model, but RPass is around 40% ahead.
Analysing the last set of values i.e. $(r, \alpha) = (64, 128)$, we could argue that the model did not run into

overfitting (though, with such high values, it might be prone to fall into that category) as the average distance is better than the previous case, while RPass has also increased. Again, when comparing with the baseline model, we can clearly see that the performance is much better now.

4 Implementation Assignment: I14

Averages	Training Memory	Number of Parameters
$(r, \alpha) = (4, 8)$	0.865 GB	7471104
$(r, \alpha) = (16, 32)$	0.0 GB	29884416
$(r, \alpha) = (64, 128)$	1.625 GB	119537664

Since the value of the parameters r and α is increased from one model to another, the following points are expected (and they are also fulfilled. This could be argued using the computed results):

- number of parameters will grow. Actually, we can see that it grows exponentially. The number of parameters for the second set of values is 4 times the initial number of parameters, whereas the third number of parameters is 4 times the second set and 16 times the first set.

- memory usage is expected to increase as the values from one set to another are increasing. However, there is an anomaly in case $r = 16$ and $\alpha = 32$. I tried in two separate Google Colab Notebooks to fine tune the model for these values and for both times, I got 0.0 GB used.

In terms of tradeoff concering the REdit and RPass, as mentioned in the previous point, it looks like the values are improving when comparing with the first model. Also, when advancing in higher number for the parameters, there is a slight downgrade for REdit that will be fixed. It will not be lowered as much as in the base model case, but it will have a smaller value. This might be because the model is learning incorrect solutions. The ability to generalize well on unseen data slightly decreases. The ups and downs of REdit and RPass could also be seen in figure 1. Equally, according to the graph, we could argue that RPass increased between models as it follows:

- from base to $(r, \alpha) = (4, 8)$ by 20%

- from $(r, \alpha) = (4, 8)$ to $(r, \alpha) = (16, 32)$ by 20%

- from $(r, \alpha) = (16, 32)$ to $(r, \alpha) = (64, 128)$ by 12%

The last increase is lower than the previous ones. This is expected because when we are closer and closer to the top / best result i.e. RPass = 100%, the difference between a good model and a better model is very little such that each small change in the parameter settings might influence its behavior.

5 Implementation Assignment: I15

Metrics generated using the multi-task model.

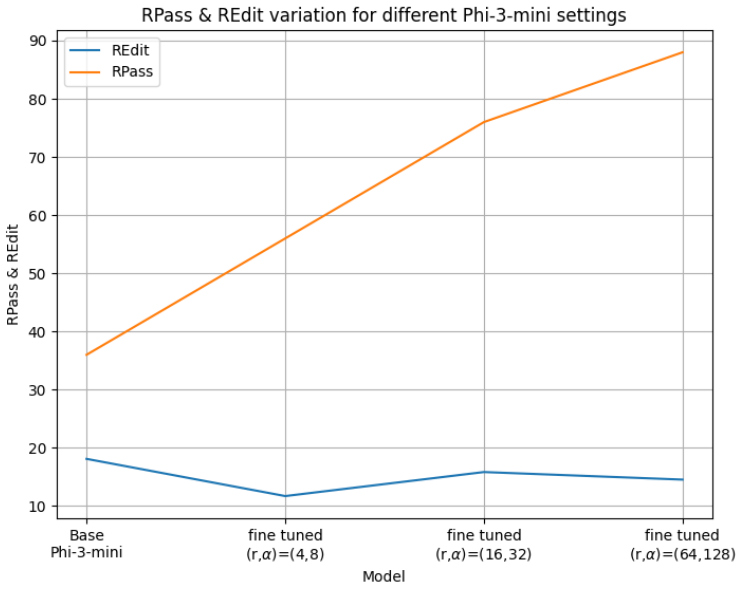

Figure 1: REdit & RPass variation (generated using `matplotlib` Python module)

Problem 1	Correct	Distance
Program 1	True	4
Program 2	True	38
Program 3	True	5
Program 4	True	19
Program 5	False	−1

Problem 2	Correct	Distance
Program 1	True	1
Program 2	True	8
Program 3	False	−1
Program 4	True	91
Program 5	True	5

Problem 3	Correct	Distance
Program 1	True	1
Program 2	False	−1
Program 3	True	11
Program 4	True	39
Program 5	True	4

Problem 4	Correct	Distance
Program 1	False	−1
Program 2	True	8
Program 3	True	5
Program 4	False	−1
Program 5	True	5

Problem 5	Correct	Distance
Program 1	True	1
Program 2	True	9
Program 3	False	−1
Program 4	True	2
Program 5	True	24

Averages	RPass (%)	REdit
Problem 1	80	16.5
Problem 2	80	26.25
Problem 3	80	13.75
Problem 4	60	6
Problem 5	80	9
Overall	76	14.736421

6 Implementation Assignment: I16 ([1])

Metrics analysis performed on results generated with the multi-task model **Phi-3-SFT-Repair_Hints_r16_alpha32_I15_I16**

Problem 1	HCorrect	HInformative	HConceal	HComprehensible	HGood
Program 1 hint	1	1	1	1	1
Program 2 hint	1	1	1	1	1
Program 3 hint	1	1	1	1	1
Program 4 hint	0	0	0	0	0
Program 5 hint	0	0	0	1	0

The hint for *Program 4* is *Consider what happens when the sequence is empty. How does your code handle this case, and what should it return according to the problem statement?*. It refers to the return statement and not to the actual problem in the code i.e. the check for empty list is not appropriately executed. Thus, the hint is neither correct nor informative. It also contains redundant information because what is written in there does not help in solving the bugs.
The hint for *Program 5* already appeared almost using the same words in a previous exercise. It says the user needs to check the case when our element is equal to the last element. This case is covered. The problem is still for the case when our element is greater than all other elements. I will consider HComprehensible = 1 because, the hint points in a slight manner to the comparison with the last element and maybe the learner might get the idea after reasoning about it for some time.

Overall for **Problem 1**, HGood = 60%.

Problem 2	HCorrect	HInformative	HConceal	HComprehensible	HGood
Program 1 hint	1	1	1	1	1
Program 2 hint	1	1	1	1	1
Program 3 hint	1	1	1	1	1
Program 4 hint	1	1	1	1	1
Program 5 hint	1	1	0	1	1

The hint for *Program 1* says to check the parameter names in all functions, but since it does not indicate a specific case, we could argue that the hint is correct and it does not reveal too much information.
The hint for *Program 3* solves at least one bug i.e. the problem with using uninitialized variables.
The hint for *Program 5* says *Check the order of your return statements in the 'contains_unique_day' function. Are you returning 'False' at the right moment?*. This is correct, but since it specifies the function where to look for the `return` statement, it reveals too much information in my opinion. Hence, HConceal = 0.

HGood = 80% in case of **Problem 2**.

Problem 3	HCorrect	HInformative	HConceal	HComprehensible	HGood
Program 1 hint	1	1	1	1	1
Program 2 hint	1	1	1	1	1
Program 3 hint	1	1	1	1	1
Program 4 hint	1	1	0	1	1
Program 5 hint	1	1	1	1	1

The hint for *Program 4* is saying directly that there might a typo when slicing: *heck the variable you're using to slice the list; it seems like there might be a typo affecting your ability to access the correct elements*. This basically gives the solution straight away as the user already know what is the issue and where it lies to.

Same as in the previous case, for **Problem 3**, HGood = 80%.

Problem 4	HCorrect	HInformative	HConceal	HComprehensible	HGood
Program 1 hint	0	0	0	0	0
Program 2 hint	1	1	0	1	1
Program 3 hint	1	1	0	1	1
Program 4 hint	1	1	1	1	1
Program 5 hint	0	0	0	0	0

The hint for *Program 1* says to check what happens when iterating over an element that we are also removing from the list. However, this is not the real issue in the code. Thus, information is incorrect and could be considered redundant.
When it comes to *Program 2*, the hint says *Remember, the 'sort' method sorts the list in place and returns 'None'. Think about how you can use this method to sort the list and then return the sorted list*. This is the solution straight away as it tells the user he cannot return `sort` directly. Thus, HConceal = 0.
The hint in case of *Program 3* is again giving the answer straight away. The hint is *Check the variable name used in your code for removing the largest element from the list. It seems like there might be a typo*. It already tells the user there is a typo and also it details the information by specifying the place in the code i.e. when removing the largest element.
The hint generated for *Program 5* is irrelevant to the actual bug. It says to check the comparison of ages of people, but the actual problem is that the `append` and `remove` at the end should be placed outside of the inner loop.

So, **Problem 4** has the following HGood percentage: HGood = 60%.

Problem 5	HCorrect	HInformative	HConceal	HComprehensible	HGood
Program 1 hint	1	1	1	1	1
Program 2 hint	1	1	1	1	1
Program 3 hint	1	1	1	1	1
Program 4 hint	1	1	1	1	1
Program 5 hint	1	1	1	1	1

The hint for *Program 2* is *Check the way you're comparing elements in the list to find the largest one. Are you comparing the right part of the elements?*. One could argue that this reveals too much information because it already points to the comparison to find the largest element. Thus, they could say HConceal = 0. However, in my opinion, indeed, it points to the comparison to find the largest element. But still, the learner needs to reason what *comparing the right part of the elements* actually means. As a conclusion, setting HConceal = 1.

In this case, all the hints are correct. Thus, for **Problem 5**, HGood = 100%. Nevertheless one could argue that some of the hints are pointing to the exact place in the code where the issue is. This is true, but, together with the second part of the hint (like it is in case *Program 2*), the learner still has to reason about a possible solution / change in the code.

All in all, for **all problems**, HGood = 84%. This high percentage is for sure coming from fine-tuning the model over both hints and repairs. Although it has to divide its optimization into two different

tasks, it seems it was pretty good at generating hints, where most of them obey the definitions of all four attributes.

7 Implementation Assignment: I17

Model	TrainingMemory	TrainingTime
Multi-task model	1.002 GB	86.53 minutes

The increase in training time in comparison to previous models is expected because the dataset used for fine-tuning is bigger this time, hence it takes more time.
In order to address the training memory, it must be mentioned that by using $r = 16$ and $\alpha = 32$, we are in the middle set with the number of parameters. So, it is normal for the specialized model with $r = 64$ and $\alpha = 128$ to have more parameters and thus, more memory used than the base line model or the first set of values. Since I have the anomally with 0.0 GB returned for the specialized model for the same parameters, unfortunately, I cannot continue further with the analysis of memory.

Now, concerning the code changes. In order to be able to fine-tune a model using combined data, firstly, I created a new method in `project_part2_ansemble_dataset.py` file called `generate_dataset_combined_repair_hint`. It is quite similar to `generate_train_dataset`. The new method grabs data from the input file and it is extracting the buggy program, the repaired code and the fix. Then, it is using the data to generate the input and output for each repair and hint. In the end, the list `combined_data_generated` contains the final information where, for each item in the list, we will make a new entry in the json that will be outputted and used for repairs and hints. Before adding data into the json, we will also shuffle the whole dataset. The concept and flow of the method is very similar to the already implemented (method came within the scripts) function `generate_train_dataset`.
Furthermore, I will address the REdit and RPass. First of all, specialized models are better than the mixed model because they are fine-tuned exclusively for one task i.e. repair or hint. By doing in this way, it is easier for them to optimize one specific task whereas for the combined model, it has to take care of two different tasks at the same time. Thus, performance might be affected.
Moreover, the multi-task model allocates optimization and memory to both repairs and hints whereas, the specialized model could concentrate on one task only. From here, we deduce that in case of multi-task, the gradient updates are shared between tasks. These very updates lead to optimization but since the model needs to divide its resources for two different jobs, the updates will not that high impact like it happens in a specialized model (where the focus is only on one job).

Acknowledgements

References

[1] Alkis Gotovos Nachiket Kotalwar and Adish Singla. *Hints-In-Browser: Benchmarking Language Models for Programming Feedback Generation*. In NeurIPS (*Datasets and BenchmarksTrack*), 2024. Paper Link: https://openreview.net/pdf?id=JRMSC08gSF.