MBPP Evaluation Analysis

This notebook summarizes results from baseline and LoRA r=8 runs using saved metrics, without loading any models. All models are run on colab through the run_on_colab code or locally on a computer with higher computing power.

We compare three model configurations:

- Baseline: Mistral-7B-Instruct (zero-shot)
- LoRA r = 8: Fine-tuned using a low-rank adapter
- LoRA r = 32: Fine-tuned using a higher-capacity adapter

All models are evaluated on Google's Mostly Basic Programming Problems (MBPP) benchmark using unit-test-based functional correctness and syntax validity metrics.

What this notebook covers

1. Loading and summarizing evaluation results
2. Comparing pass@1 rates and syntax correctness rates across models
3. Visualizing performance differences between LoRA ranks
4. Analyzing why higher LoRA rank does not necessarily yield better results
5. Inspecting qualitative examples of generated code

This notebook serves as both an analysis and a lightweight demonstration of how LoRA fine-tuning affects code generation quality in practice.

Comparing Baseline to LoRA

In this section, we look at how much LoRA fine-tuning improves Python code generation by comparing a zero-shot baseline model against a LoRA-adapted model with rank r = 8. Both models are evaluated on Google's Mostly Basic Programming Problems (MBPP) benchmark using the same prompts and the same unit-test-based evaluation.

What we measure

We focus on two simple but important metrics: - Pass rate: the fraction of tasks where the generated code passes all provided unit tests. - Syntax rate: the fraction of outputs that are valid Python code and run without syntax errors.

```
import json, os, ast
from typing import Dict, List, Tuple

# Paths
# BASELINE_RESULTS = "artifacts/metrics/baseline_mbpp_results.json"
BASELINE_GENERATIONS = "artifacts/metrics/baseline_generations.jsonl"
MBPP_TEST = "data/processed/mbpp_test.jsonl"
R8_RESULTS = "artifacts/metrics/mistral7b-code-r8-mbpp_results.json"


def load_jsonl(path: str) -> List[Dict]:
    items: List[Dict] = []
    with open(path, "r", encoding="utf-8") as f:
        for line in f:
            line = line.strip()
            if not line:
                continue
            items.append(json.loads(line))
    return items


def safe_syntax_ok(code: str) -> bool:
    try:
        ast.parse(code)
        return True
    except Exception:
        return False


def run_tests_on_code(code: str, tests: List[str]) -> Tuple[bool, List[str]]:
    # WARNING: executes code/tests; use only in trusted environments
    g: Dict = {}
    try:
        exec(code, g, g)  # noqa: S102
    except Exception as e:
        return False, [f"Execution error: {type(e).__name__}: {e}"]
    errors: List[str] = []
    for t in tests:
        try:
```

```
            exec(t, g, g)  # noqa: S102
        except Exception as e:
            errors.append(f"Test failed: {t} -> {type(e).__name__}: {e}")
    return (len(errors) == 0), errors
```

```
# Load r=8 results
r8 = json.load(open(R8_RESULTS, "r", encoding="utf-8")) if os.path.exists(R8_RESULTS) else No
if r8:
    print("LoRA r=8 summary:", r8["summary"])  # contains total, syntax_rate, pass_rate
else:
    print("LoRA r=8 results not found at:", R8_RESULTS)
```

```
LoRA r=8 summary: {'total': 500, 'syntax_ok': 495, 'syntax_rate': 0.99, 'pass': 136, 'pass_ra
```

```
# Load or compute baseline metrics
baseline = None
if os.path.exists(BASELINE_RESULTS):
    try:
        baseline = json.load(open(BASELINE_RESULTS, "r", encoding="utf-8"))
        print("Baseline summary (from results):", baseline["summary"])
    except Exception as e:
        print("Could not read baseline results:", e)

if baseline is None and os.path.exists(BASELINE_GENERATIONS) and os.path.exists(MBPP_TEST):
    print("Baseline summary not found; computing from generations + MBPP tests (this may take
    gens = load_jsonl(BASELINE_GENERATIONS)
    tests = {ex.get("task_id"): (ex.get("tests") or []) for ex in load_jsonl(MBPP_TEST)}
    total = len(gens)
    num_syntax_ok = 0
    num_pass = 0
    for idx, g in enumerate(gens, start=1):
        code = g.get("generated", "").strip()
        ok = safe_syntax_ok(code)
        if ok:
            num_syntax_ok += 1
        task_id = g.get("task_id")
        tlist = tests.get(task_id, [])
        passed = False
        if tlist:
```

```
            p, _ = run_tests_on_code(code, tlist)
            passed = p
        if passed:
            num_pass += 1
        if idx % 50 == 0 or idx == total:
            print(f"[baseline] processed {idx}/{total}")
    baseline_summary = {
        "total": total,
        "syntax_ok": num_syntax_ok,
        "syntax_rate": num_syntax_ok / max(1, total),
        "pass": num_pass,
        "pass_rate": num_pass / max(1, total),
        "model": "mistralai/Mistral-7B-Instruct-v0.2",
        "lora_dir": None,
    }
    print("Baseline summary (computed):", baseline_summary)
else:
    if baseline is None:
        print("Baseline generations or MBPP tests not found; skipping baseline computation.")
```

```
Baseline summary (from results): {'total': 500, 'syntax_ok': 486, 'syntax_rate': 0.972, 'pass
```

```
# Comparison
r8_summary = r8["summary"] if r8 else None
baseline_summary = baseline["summary"] if isinstance(baseline, dict) and "summary" in baselin
    locals().get("baseline_summary") if "baseline_summary" in locals() else None
)

if r8_summary and baseline_summary:
    print("Baseline pass_rate:", round(baseline_summary["pass_rate"], 3), "syntax_rate:", rou
    print("LoRA r=8 pass_rate:", round(r8_summary["pass_rate"], 3), "syntax_rate:", round(r8_
else:
    print("Not enough data to compare both baselines and r=8.")
```

```
Baseline pass_rate: 0.022 syntax_rate: 0.972
LoRA r=8 pass_rate: 0.272 syntax_rate: 0.99
```

Analysis

Baseline Evaluation

The zero-shot baseline model already does a decent job producing valid-looking Python. About 97% of its outputs are syntactically correct. However, this masks a deeper issue: only 2.2% of generated solutions actually pass the unit tests. In practice, the baseline often produces incomplete logic, misses edge cases, or misunderstands the task requirements.

Since a precomputed baseline results file was unavailable, these metrics are computed directly by re-running the MBPP unit tests on saved baseline generations. This ensures that the baseline is evaluated using the same procedure as the fine-tuned models.

LoRA r = 8 Evaluation

After fine-tuning with LoRA at rank r = 8, performance improves dramatically. The model now passes 27.2% of MBPP tasks, representing more than a twelve-fold increase in functional correctness. Syntax correctness also improves slightly to 99%, indicating that fine-tuning does not introduce instability or malformed code.

Qualitative Examinations

To better understand the performance gap between the baseline model and the LoRA-fine-tuned model (r = 8), we examined concrete examples where tests failed. This qualitative analysis reveals not just how often the models fail, but why.

```python
# Sample successes and failures from r=8
if r8 and "results" in r8:
    results = r8["results"]
    passed = [r for r in results if r.get("passed")]
    failed = [r for r in results if not r.get("passed")]
    print("Examples - Passed:")
    for ex in passed[:3]:
        print("- task", ex.get("task_id"), "|", ex.get("instruction")[:80])
        print((ex.get("generated") or "").split("\n")[0][:120], "...\n")
    print("Examples - Failed:")
    for ex in failed[:3]:
        print("- task", ex.get("task_id"), "|", ex.get("instruction")[:80])
        errs = ex.get("errors") or []
        print("Error:", errs[0] if errs else "(no details)")
else:
    print("r=8 results not found or missing 'results' field.")
```

```
Examples - Passed:
- task 12 | Write a function to sort a given matrix in ascending order according to the sum
def sort_matrix(matrix): ...

- task 17 | Write a function to find the perimeter of a square.
```

```
def square_perimeter(side): ...

- task 18 | Write a function to remove characters from the first string which are present in
def remove_dirty_chars(string1, string2): ...

Examples - Failed:
- task 11 | Write a python function to remove first and last occurrence of a given character
Error: Test failed: assert remove_Occ("hello","l") == "heo" -> AssertionError:
- task 13 | Write a function to count the most common words in a dictionary.
Error: Test failed: assert count_common(['red','green','black','pink','black','white','black
- task 14 | Write a python function to find the volume of a triangular prism.
Error: Test failed: assert find_Volume(10,8,6) == 240 -> AssertionError:
```

Common failure modes for LoRA r = 8

Although the LoRA-tuned model significantly improves overall pass rate, it still fails on a subset of tasks. Most of these failures fall into a few recurring patterns:

- Logical edge cases: In several tasks (e.g., computing volumes, combinatorics, or counting values), the model produces a reasonable-looking implementation that misses edge cases or slightly misinterprets the problem constraints.
- Incorrect return values: Some functions compute the correct intermediate logic but return the wrong value or format, leading to assertion failures.
- Overly simplified logic: In tasks involving counting, aggregation, or sorting, the model sometimes opts for a simplified approach that works for common inputs but fails under stricter test conditions.

```python
# Print test cases where the trained model (r=8) passed and the baseline failed

if r8 and "results" in r8 and baseline and "results" in baseline:
    # Index baseline results by task_id for fast lookup
    baseline_results_by_task = {r.get("task_id"): r for r in baseline["results"]}
    r8_results = r8["results"]

    cases_passed_r8_failed_baseline = []
    for r8_ex in r8_results:
        task_id = r8_ex.get("task_id")
        baseline_ex = baseline_results_by_task.get(task_id)
        if not baseline_ex:
            continue
        if r8_ex.get("passed") and not baseline_ex.get("passed"):
            cases_passed_r8_failed_baseline.append((r8_ex, baseline_ex))
```

```
    print(f"\nTest cases where LoRA r=8 PASSED but baseline FAILED ({len(cases_passed_r8_fai
    for i, (r8_ex, baseline_ex) in enumerate(cases_passed_r8_failed_baseline[:40]):  # print
        print(f"--- Example {i+1} ---")
        print("Task ID:", r8_ex.get("task_id"))
        instr = r8_ex.get("instruction", "")
        print("Instruction:", instr[:200] + ("..." if len(instr) > 200 else ""))
        print("\nLoRA r=8 generated:\n", (r8_ex.get("generated") or "")[:400], "\n")
        baseline_gen = baseline_ex.get("generated") or ""
        print("Baseline generated:\n", baseline_gen[:400], "\n")
        baseline_errs = baseline_ex.get("errors") or []
        err_msg = baseline_errs[0] if baseline_errs else "(no error details)"
        print("Baseline error:", err_msg)
        print("-" * 80)
    if not cases_passed_r8_failed_baseline:
        print("No cases found where r=8 passed and baseline failed.")
else:
    print("Cannot compute delta cases: missing results in r=8 or baseline.")
```

```
Test cases where LoRA r=8 PASSED but baseline FAILED (129 cases):
--- Example 1 ---
Task ID: 12
Instruction: Write a function to sort a given matrix in ascending order according to the sum

LoRA r=8 generated:
 def sort_matrix(matrix):
    matrix.sort(key=lambda row: sum(row))
    return matrix

Baseline generated:
 import numpy as np

def sort_matrix_by_row_sum(matrix):
    """
    Sort a given matrix in ascending order according to the sum of its rows.

    :param matrix: A 2D NumPy array.
    :return: A NumPy array with the rows sorted in ascending order based on their sums.
    """

    # Calculate the sum of each row and store it in a new array
```

```
    row_sums = np.sum(matrix, axis=1)

    # Use the argsort

Baseline error: Test failed: assert sort_matrix([[1, 2, 3], [2, 4, 5], [1, 1, 1]])==[[1, 1, 1
--------------------------------------------------------------------------------
--- Example 2 ---
Task ID: 18
Instruction: Write a function to remove characters from the first string which are present in

LoRA r=8 generated:
 def remove_dirty_chars(string1, string2):
    return ''.join([char for char in string1 if char not in string2])

Baseline generated:
 def remove_characters(string_one, string_two):
    """
    :param string_one: The first string from which characters need to be removed.
    :param string_two: The second string containing the characters to be removed.
    :return: A new string with the characters from string_one that are not in string_two remo
    """
    return ''.join(c for c in string_one if c not in string_two)

Baseline error: Test failed: assert remove_dirty_chars("probasscurve", "pros") == 'bacuve' -
--------------------------------------------------------------------------------
--- Example 3 ---
Task ID: 19
Instruction: Write a function to find whether a given array of integers contains any duplicat

LoRA r=8 generated:
 def test_duplicate(arr):
    seen = set()
    for num in arr:
        if num in seen:
            return True
        seen.add(num)
    return False

Baseline generated:
 def find_duplicate(arr):
    """
    :param arr: List of integers
    :return: True if there is a duplicate in the list, False otherwise
```

```
    """
    s = set(arr)
    return len(arr) != len(s)

Baseline error: Test failed: assert test_duplicate(([1,2,3,4,5]))==False -> NameError: name
--------------------------------------------------------------------------------
--- Example 4 ---
Task ID: 22
Instruction: Write a function to find the first duplicate element in a given array of integer

LoRA r=8 generated:
 def find_first_duplicate(arr):
    seen = set()
    for num in arr:
        if num in seen:
            return num
        else:
            seen.add(num)
    return -1

Baseline generated:
 def find_first_duplicate(numbers):
    """
    Find the first duplicate element in a given list of integers.

    :param numbers: List of integers
    :return: The first duplicate element
    """

    # Use a set to store unique elements
    unique_elements = set(numbers)

    # If the length of the list is equal to the length of the set, then all elements are unic
    if len(numbers) == len(lis

Baseline error: Test failed: assert find_first_duplicate([1, 2, 3, 4])==-1 -> ValueError: No
--------------------------------------------------------------------------------
--- Example 5 ---
Task ID: 23
Instruction: Write a python function to find the maximum sum of elements of list in a list o

LoRA r=8 generated:
 def maximum_Sum(lists):
```

```
    max_sum = 0
    for list in lists:
        sum = 0
        for element in list:
            sum += element
        max_sum = max(max_sum, sum)
    return max_sum

Baseline generated:
 def max_sum_list_of_lists(list_of_lists):
    """
    Find the maximum sum of elements in a list of lists.

    :param list_of_lists: List of lists.
    :return: Maximum sum.
    """
    max_sum = float('-inf')
    for sub_list in list_of_lists:
        current_sum = sum(sub_list)
        if current_sum > max_sum:
            max_sum = current_sum
    return max_sum

Baseline error: Test failed: assert maximum_Sum([[1,2,3],[4,5,6],[10,11,12],[7,8,9]]) == 33 -
--------------------------------------------------------------------------------
--- Example 6 ---
Task ID: 28
Instruction: Write a python function to find binomial co-efficient.

LoRA r=8 generated:
 def binomial_Coeff(n, k):
    if k == 0 or k == n:
        return 1
    else:
        return binomial_Coeff(n-1, k-1) + binomial_Coeff(n-1, k)

Baseline generated:
 def binomial_coefficient(n, k):
    """
    Calculate the binomial coefficient (nCk).

    :param int n: The total number of items.
    :param int k: The number of items to choose.
```

```
    :return: The binomial coefficient.
    """

    # Memoization table to store previously calculated values
    memo = {}

    def _binomial_coefficient(n, k):
        if (n, k) in memo:
            return memo[(n, k)

Baseline error: Test failed: assert binomial_Coeff(5,2) == 10 -> NameError: name 'binomial_Co
--------------------------------------------------------------------------------
--- Example 7 ---
Task ID: 41
Instruction: Write a function to filter even numbers using lambda function.

LoRA r=8 generated:
 def filter_evennumbers(numbers):
    return list(filter(lambda x: x % 2 == 0, numbers))

Baseline generated:
 def filter_even_numbers(input_list):
    """
    Filter even numbers from the given list using a lambda function.

    :param input_list: List of numbers
    :return: List of even numbers
    """
    return list(filter(lambda x: x % 2 == 0, input_list))

Baseline error: Test failed: assert filter_evennumbers([1, 2, 3, 4, 5, 6, 7, 8, 9, 10])==[2,
--------------------------------------------------------------------------------
--- Example 8 ---
Task ID: 46
Instruction: Write a python function to determine whether all the numbers are different from

LoRA r=8 generated:
 def test_distinct(numbers):
    return len(set(numbers)) == len(numbers)

Baseline generated:
 def unique_numbers(numbers):
    """
```

```
    :param numbers: A list of numbers
    :return: True if all numbers in the list are unique, False otherwise
    """
    return len(numbers) == len(set(numbers))

Baseline error: Test failed: assert test_distinct([1,5,7,9]) == True -> NameError: name 'test
--------------------------------------------------------------------------------
--- Example 9 ---
Task ID: 49
Instruction: Write a function to extract every first or specified element from a given two-di

LoRA r=8 generated:
 def specified_element(lst, index):
    result = []
    for row in lst:
        result.append(row[index])
    return result

Baseline generated:
 def extract_first_or_specified(lst, index=0):
    """
    Extracts the index-th element from each sub-list in a given two-dimensional list.

    :param lst: A two-dimensional list.
    :param index: The index of the element to extract from each sub-list. Default is 0, which
    :return: A list containing the extracted elements.
    """
    return [sublist[index] for sublis

Baseline error: Test failed: assert specified_element([[1, 2, 3, 2], [4, 5, 6, 2], [7, 1, 9,
--------------------------------------------------------------------------------
--- Example 10 ---
Task ID: 56
Instruction: Write a python function to check if a given number is one less than twice its re

LoRA r=8 generated:
 def check(num):
    return num == 2 * int(str(num)[::-1]) - 1

Baseline generated:
 def is_half_reverse(num):
    """
    Check if a given number is one less than twice its reverse.
```

```
    :param num: An integer
    :return: A boolean value
    """

    # Reverse the number
    reversed_num = int(str(num)[::-1])

    # Check if the number is one less than twice the reversed number
    return num == reversed_num * 2 - 1

Baseline error: Test failed: assert check(70) == False -> NameError: name 'check' is not defi
--------------------------------------------------------------------------------
--- Example 11 ---
Task ID: 58
Instruction: Write a python function to check whether the given two integers have opposite si

LoRA r=8 generated:
 def opposite_Signs(a, b):
    if a < 0 and b > 0 or a > 0 and b < 0:
        return True
    else:
        return False

Baseline generated:
 def have_opposite_signs(num1, num2):
    """
    :param num1: The first integer
    :param num2: The second integer
    :return: True if the integers have opposite signs, False otherwise
    """
    return num1 < 0 and num2 > 0 or num1 > 0 and num2 < 0

Baseline error: Test failed: assert opposite_Signs(1,-2) == True -> NameError: name 'opposite
--------------------------------------------------------------------------------
--- Example 12 ---
Task ID: 62
Instruction: Write a python function to find smallest number in a list.

LoRA r=8 generated:
 def smallest_num(nums):
    min_num = nums[0]
    for num in nums:
```

```
        if num < min_num:
            min_num = num
    return min_num

Baseline generated:
 def find_smallest(numbers):
    """
    Finds the smallest number in the given list.

    :param numbers: List of numbers
    :return: The smallest number in the list
    """
    return min(numbers)

Baseline error: Test failed: assert smallest_num([10, 20, 1, 45, 99]) == 1 -> NameError: name
--------------------------------------------------------------------------------
--- Example 13 ---
Task ID: 64
Instruction: Write a function to sort a list of tuples using lambda.

LoRA r=8 generated:
 def subject_marks(marks):
    marks.sort(key=lambda x: x[1])
    return marks

Baseline generated:
 def sort_tuples_with_lambda(tuples_list):
    """
    Sorts a list of tuples using a lambda function as the key.

    :param tuples_list: List of tuples to be sorted.
    :return: Sorted list of tuples.
    """
    sorted_tuples = sorted(tuples_list, key=lambda x: x[1])
    return sorted_tuples

Baseline error: Test failed: assert subject_marks([('English', 88), ('Science', 90), ('Maths
--------------------------------------------------------------------------------
--- Example 14 ---
Task ID: 66
Instruction: Write a python function to count positive numbers in a list.

LoRA r=8 generated:
```

```
 def pos_count(nums):
    count = 0
    for num in nums:
        if num > 0:
            count += 1
    return count

Baseline generated:
 def count_positives(numbers):
    """
    :param numbers: list of numbers
    :return: int, the number of positive numbers in the list
    """
    return len(list(filter(lambda x: x > 0, numbers)))

Baseline error: Test failed: assert pos_count([1,-2,3,-4]) == 2 -> NameError: name 'pos_count
--------------------------------------------------------------------------------
--- Example 15 ---
Task ID: 68
Instruction: Write a python function to check whether the given array is monotonic or not.

LoRA r=8 generated:
 def is_Monotonic(arr):
    # Check if the array is empty
    if len(arr) == 0:
        return True

    # Check if the array is increasing
    is_increasing = True
    for i in range(1, len(arr)):
        if arr[i-1] > arr[i]:
            is_increasing = False
            break

    # Check if the array is decreasing
    is_decreasing = True
    for i in range(1, len(arr)):
        if arr[i-1] < a

Baseline generated:
 def is_monotonic(arr):
    """
    :param arr: List or tuple
```

```
    :return: bool
    """
    if len(arr) < 2:
        return True

    is_increasing = all(arr[i] <= arr[i+1] for i in range(len(arr)-1))
    is_decreasing = all(arr[i] >= arr[i+1] for i in range(len(arr)-1))

    return is_increasing or is_decreasing

Baseline error: Test failed: assert is_Monotonic([6, 5, 4, 4]) == True -> NameError: name 'is
--------------------------------------------------------------------------------
--- Example 16 ---
Task ID: 69
Instruction: Write a function to check whether a list contains the given sublist or not.

LoRA r=8 generated:
 def is_sublist(lst, sublist):
    for i in range(len(lst)):
        if lst[i:i+len(sublist)] == sublist:
            return True
    return False

Baseline generated:
 def list_contains_sublist(main_list, sublist):
    """
    :param main_list: The list to search in.
    :param sublist: The sublist to search for.
    :return: True if sublist is in main_list, False otherwise.
    """
    return sublist in main_list

Baseline error: Test failed: assert is_sublist([2,4,3,5,7],[3,7])==False -> NameError: name
--------------------------------------------------------------------------------
--- Example 17 ---
Task ID: 71
Instruction: Write a function to sort a list of elements using comb sort.

LoRA r=8 generated:
 def comb_sort(arr):
    n = len(arr)
    gap = n // 2
    while gap > 0:
```

```
        for i in range(n - gap):
            j = i + gap
            if arr[i] > arr[j]:
                arr[i], arr[j] = arr[j], arr[i]
        gap //= 2
    for i in range(n - 1):
        if arr[i] > arr[i + 1]:
            arr[i], arr[i + 1] = arr[i + 1], arr[i]
    return arr

Baseline generated:
 def comb_sort(arr):
    n = len(arr)
    gap = int(n * 0.5)

    while gap > 0:
        for i in range(n):
            j = i
            while j >= 0 and arr[j] > arr[j + gap]:
                arr[j], arr[j + gap] = arr[j + gap], arr[j]
                j -= gap

        gap = int(gap / 1.3)

arr = [64, 34, 23, 1, 55, 21, 32, 41, 15]
comb_sort(arr)
print(arr)

Baseline error: Execution error: IndexError: list index out of range
--------------------------------------------------------------------------------
--- Example 18 ---
Task ID: 78
Instruction: Write a python function to find number of integers with odd number of set bits.

LoRA r=8 generated:
 def count_With_Odd_SetBits(n):
    count = 0
    for i in range(n):
        count += bin(i).count('1') % 2 == 1
    return count

Baseline generated:
 def count_odd_set_bits(start, end):
```

```
    """
    :param start: The start of the range
    :param end: The end of the range
    :return: The number of integers in the range with an odd number of set bits
    """
    count = 0
    for i in range(start, end + 1):
        bit_count = 0
        while i > 0:
            i, bit = divmod(i, 2)
            bit_count += bit
        if bit_count % 2 == 1:


Baseline error: Test failed: assert count_With_Odd_SetBits(5) == 3 -> NameError: name 'count
--------------------------------------------------------------------------------
--- Example 19 ---
Task ID: 79
Instruction: Write a python function to check whether the length of the word is odd or not.

LoRA r=8 generated:
 def word_len(word):
    return len(word) % 2 == 1

Baseline generated:
 def is_odd_length(word: str) -> bool:
    """
    Check if the given word has an odd number of characters.

    :param word: The word to check.
    :return: True if the word has an odd number of characters, False otherwise.
    """
    return len(word) % 2 != 0

Baseline error: Test failed: assert word_len("Hadoop") == False -> NameError: name 'word_len
--------------------------------------------------------------------------------
--- Example 20 ---
Task ID: 89
Instruction: Write a function to find the closest smaller number than n.

LoRA r=8 generated:
 def closest_num(n):
    """
```

```
    Find the closest smaller number than n
    """
    # Initialize the closest smaller number
    closest = 0
    # Iterate through the numbers from 0 to n
    for i in range(n+1):
        # If the current number is smaller than n
        if i < n:
            # Update the closest number if the current number is closer
            if abs(n - i) < abs(n - closest):


Baseline generated:
 def find_closest_smaller(numbers, n):
    """
    Find the closest smaller number in the given list to the number n.

    :param numbers: A list of numbers.
    :param n: The number to find the closest smaller number for.
    :return: The closest smaller number in the list.
    """

    # Filter out numbers smaller than n from the list
    smaller_numbers = [x for x in numbers if x < n]

    # If

Baseline error: Test failed: assert closest_num(11) == 10 -> NameError: name 'closest_num' is
--------------------------------------------------------------------------------
--- Example 21 ---
Task ID: 90
Instruction: Write a python function to find the length of the longest word.

LoRA r=8 generated:
 def len_log(words):
    max_length = 0
    for word in words:
        if len(word) > max_length:
            max_length = len(word)
    return max_length

Baseline generated:
 def find_longest_word_length(words):
```

```
    """
    Finds the length of the longest word in the given list of words.

    :param words: list of words
    :return: int, the length of the longest word
    """
    return max(len(word) for word in words)

Baseline error: Test failed: assert len_log(["python","PHP","bigdata"]) == 7 -> NameError: na
--------------------------------------------------------------------------------
--- Example 22 ---
Task ID: 94
Instruction: Write a function to extract the index minimum value record from the given tuples

LoRA r=8 generated:
 def index_minimum(tuples):
    min_val = min(tuples, key=lambda x: x[1])
    return min_val[0]

Baseline generated:
 def min_index(tuples):
    """
    :param tuples: A list of tuples, where each tuple contains a value.
    :return: A tuple containing the index and the minimum value.
    """
    min_value, index = min(tuples, key=lambda x: x[0])
    return (tuples.index(min_value), min_value)

Baseline error: Test failed: assert index_minimum([('Rash', 143), ('Manjeet', 200), ('Varsha
--------------------------------------------------------------------------------
--- Example 23 ---
Task ID: 96
Instruction: Write a python function to find the number of divisors of a given integer.

LoRA r=8 generated:
 def divisor(n):
    count = 0
    for i in range(1, n+1):
        if n % i == 0:
            count += 1
    return count

Baseline generated:
```

```
 import math

def divisors_count(n):
    """
    Returns the number of divisors of the given integer.
    """
    return int(math.pow(math.gcd(n, 1), 2))

Baseline error: Test failed: assert divisor(15) == 4  -> NameError: name 'divisor' is not de
--------------------------------------------------------------------------------
--- Example 24 ---
Task ID: 97
Instruction: Write a function to find frequency count of list of lists.

LoRA r=8 generated:
 def frequency_lists(lists):
    freq_dict = {}
    for list in lists:
        for item in list:
            if item not in freq_dict:
                freq_dict[item] = 1
            else:
                freq_dict[item] += 1
    return freq_dict

Baseline generated:
 from collections import Counter

def frequency_count(lst_of_lists):
    """
    This function takes a list of lists as an argument and returns a dictionary with the fre
    """
    counter = Counter(itertools.chain.from_iterable(lst_of_lists))
    return counter.items()

Baseline error: Test failed: assert frequency_lists([[1, 2, 3, 2], [4, 5, 6, 2], [7, 8, 9, 5
--------------------------------------------------------------------------------
--- Example 25 ---
Task ID: 99
Instruction: Write a function to convert the given decimal number to its binary equivalent.

LoRA r=8 generated:
 def decimal_to_binary(num):
```

```
    return bin(num).replace("0b", "")

Baseline generated:
 def dec_to_bin(decimal):
    """
    This function converts a decimal number to its binary equivalent.

    :param decimal: The decimal number to be converted.
    :type decimal: int or float
    :return: The binary equivalent of the decimal number.
    :rtype: str
    """

    # Check if input is valid
    if not isinstance(decimal, (int, float)):
        raise TypeError("Input must be a number")

Baseline error: Test failed: assert decimal_to_binary(8) == '1000' -> NameError: name 'decima
--------------------------------------------------------------------------------
--- Example 26 ---
Task ID: 104
Instruction: Write a function to sort each sublist of strings in a given list of lists using

LoRA r=8 generated:
 def sort_sublists(lst):
    return [sorted(sublist, key=lambda x: x[0]) for sublist in lst]

Baseline generated:
 def sort_sublists(input_list):
    """
    This function takes a list of lists as an argument and sorts each sublist using a lambda
    """
    return [list(map(lambda x: sorted(x), input_list))]

# Example usage:
input_list = [['apple', 'banana', 'orange'], ['grape', 'kiwi', 'mango'], ['peach', 'pear', 'p
sorted_list_of_lists = sort_sublists(input_list)
print(sorted_list_of_l

Baseline error: Test failed: assert sort_sublists((["green", "orange"], ["black", "white"],
--------------------------------------------------------------------------------
--- Example 27 ---
Task ID: 105
```

```
Instruction: Write a python function to count true booleans in the given list.

LoRA r=8 generated:
 def count(lst):
    return sum(lst)

Baseline generated:
 def count_true_booleans(boolean_list):
    """
    :param boolean_list: List of boolean values
    :return: Number of True values in the list
    """
    return sum(boolean_list)

Baseline error: Test failed: assert count([True,False,True]) == 2 -> NameError: name 'count'
--------------------------------------------------------------------------------
--- Example 28 ---
Task ID: 113
Instruction: Write a function to check if a string represents an integer or not.

LoRA r=8 generated:
 def check_integer(string):
    try:
        int(string)
        return True
    except ValueError:
        return False

Baseline generated:
 def is_valid_integer(string):
    try:
        int(string)
        return True
    except ValueError:
        return False

Baseline error: Test failed: assert check_integer("python")==False -> NameError: name 'check
--------------------------------------------------------------------------------
--- Example 29 ---
Task ID: 115
Instruction: Write a function to check whether all dictionaries in a list are empty or not.

LoRA r=8 generated:
```

```
 def empty_dit(dic_list):
    for dic in dic_list:
        if dic:
            return False
    return True

Baseline generated:
 def all_dicts_empty(dicts):
    """
    :param dicts: A list of dictionaries
    :return: True if all dictionaries in the list are empty, False otherwise
    """
    return all(len(d) == 0 for d in dicts)

Baseline error: Test failed: assert empty_dit([{},{},{}])==True -> NameError: name 'empty_dit
--------------------------------------------------------------------------------
--- Example 30 ---
Task ID: 116
Instruction: Write a function to convert a given tuple of positive integers into an integer.

LoRA r=8 generated:
 def tuple_to_int(t):
    return int(''.join(str(x) for x in t))

Baseline generated:
 def tuple_to_integer(tuple):
    """
    Converts a given tuple of positive integers into a single integer.

    :param tuple: A tuple of positive integers
    :return: An integer, which is the sum of all the integers in the given tuple
    """
    return sum(tuple)

Baseline error: Test failed: assert tuple_to_int((1,2,3))==123 -> NameError: name 'tuple_to_i
--------------------------------------------------------------------------------
--- Example 31 ---
Task ID: 118
Instruction: [link text](https:// [link text](https:// [link text](https://)))write a functio

LoRA r=8 generated:
 def string_to_list(string):
    return string.split()
```

```
Baseline generated:
 def string_to_list(input_string):
    return [char for char in input_string]

Baseline error: Test failed: assert string_to_list("python programming")==['python','programm
--------------------------------------------------------------------------------
--- Example 32 ---
Task ID: 126
Instruction: Write a python function to find the sum of common divisors of two given numbers

LoRA r=8 generated:
 def sum(a, b):
    sum = 0
    for i in range(1, min(a, b) + 1):
        if a % i == 0 and b % i == 0:
            sum += i
    return sum

Baseline generated:
 def gcd(a, b):
    while b:
        a, b = b, a % b
    return a

def sum_of_common_divisors(num1, num2):
    gcd_value = gcd(num1, num2)
    return (num1 // gcd_value + num2 // gcd_value) * gcd_value

Baseline error: Test failed: assert sum(10,15) == 6 -> TypeError: 'int' object is not iterabl
--------------------------------------------------------------------------------
--- Example 33 ---
Task ID: 127
Instruction: Write a function to multiply two integers without using the * operator in pytho

LoRA r=8 generated:
 def multiply_int(x, y):
    result = 0
    for i in range(abs(y)):
        result += x
    return result if y > 0 else -result

Baseline generated:
```

```
 def multiply(x, y):
    if y == 0:
        return 0
    else:
        return x + multiply(x, y - 1)

# Test the function
x = 5
y = 3
result = multiply(x, y)
print(result)  # Output: 15

Baseline error: Test failed: assert multiply_int(10,20)==200 -> NameError: name 'multiply_int
------------------------------------------------------------------------------
--- Example 34 ---
Task ID: 133
Instruction: Write a function to calculate the sum of the negative numbers of a given list o

LoRA r=8 generated:
 def sum_negativenum(numbers):
    return sum(x for x in numbers if x < 0)

Baseline generated:
 def sum_negatives(numbers):
    return sum(filter(lambda x: x < 0, numbers))

Baseline error: Test failed: assert sum_negativenum([2, 4, -6, -9, 11, -12, 14, -5, 17])==-3
------------------------------------------------------------------------------
--- Example 35 ---
Task ID: 167
Instruction: Write a python function to find smallest power of 2 greater than or equal to n.

LoRA r=8 generated:
 def next_Power_Of_2(n):
    power = 0
    while (n > 0):
        power += 1
        n >>= 1
    return (1 << power)

Baseline generated:
 def smallest_power_of_two(n):
    """
```

```
    Returns the smallest power of 2 greater than or equal to n.
    """
    power = 0
    while (2 ** power < n):
        power += 1
    return power

Baseline error: Test failed: assert next_Power_Of_2(0) == 1 -> NameError: name 'next_Power_O
--------------------------------------------------------------------------------
--- Example 36 ---
Task ID: 168
Instruction: Write a python function to find the frequency of a number in a given array.

LoRA r=8 generated:
 def frequency(arr, num):
    count = 0
    for i in arr:
        if i == num:
            count += 1
    return count

Baseline generated:
 def count_frequency(num, arr):
    """
    :param num: The number to find the frequency of.
    :param arr: The list in which to search for the number.
    :return: The frequency of the number in the list.
    """
    return len(list(filter(lambda x: x == num, arr)))

Baseline error: Test failed: assert frequency([1,2,3],4) == 0 -> NameError: name 'frequency'
--------------------------------------------------------------------------------
--- Example 37 ---
Task ID: 170
Instruction: Write a function to find sum of the numbers in a list between the indices of a s

LoRA r=8 generated:
 def sum_range_list(list, start, end):
    sum = 0
    for i in range(start, end + 1):
        sum += list[i]
    return sum
```

```
Baseline generated:
 def sum_list_slice(my_list, start_index, end_index):
    """
    Calculate the sum of numbers in a list between the specified range of indices.

    :param my_list: List of numbers
    :param start_index: Index of the first number to include in the sum
    :param end_index: Index of the first number after the last number to include in the sum
    :return: The sum of numbers in the specified range


Baseline error: Test failed: assert sum_range_list( [2,1,5,6,8,3,4,9,10,11,8,12],8,10)==29 -
--------------------------------------------------------------------------------
--- Example 38 ---
Task ID: 171
Instruction: Write a function to find the perimeter of a pentagon.

LoRA r=8 generated:
 def perimeter_pentagon(side_length):
    return 5 * side_length

Baseline generated:
 def perimeter_of_pentagon(side_length):
    """
    Calculate the perimeter of a regular pentagon.

    :param side_length: The length of one side of the pentagon.
    :type side_length: float
    :return: The perimeter of the pentagon.
    :rtype: float
    """
    perimeter = 5 * side_length
    return perimeter

Baseline error: Test failed: assert perimeter_pentagon(5)==25 -> NameError: name 'perimeter_
--------------------------------------------------------------------------------
--- Example 39 ---
Task ID: 172
Instruction: Write a function to find the occurence of characters 'std' in the given string

LoRA r=8 generated:
 def count_occurance(string):
    count = 0
```

```
    for i in range(len(string)-2):
        if string[i:i+3] == 'std':
            count += 1
    return count

Baseline generated:
 def count_std(lst):
"""
Function to find the number of occurrences of 'std' in a list of strings.

:param lst: list of strings
:return: int, number of occurrences of 'std' in the list
"""
return sum(1 for string in lst if 'std' in string)

Baseline error: Execution error: IndentationError: expected an indented block after function
--------------------------------------------------------------------------------
--- Example 40 ---
Task ID: 173
Instruction: Write a function to remove everything except alphanumeric characters from a str

LoRA r=8 generated:
 def remove_splchar(string):
    return ''.join(c for c in string if c.isalnum())

Baseline generated:
 import re

def remove_non_alphanumeric(string):
    """
    This function removes all non-alphanumeric characters from a given string.

    :param string: The input string
    :return: The string with only alphanumeric characters
    """
    return re.sub(r'\W+', '', string)

Baseline error: Test failed: assert remove_splchar('python  @#&^%$*program123')==('pythonprog
--------------------------------------------------------------------------------
```

Discussion

The contrast between the baseline and the LoRA-finetuned model highlights an important

point: producing syntactically valid code is not the hard part. Understanding the task and translating it into correct logic is. LoRA fine-tuning helps the model bridge this gap, turning mostly-plausible code into solutions that actually work.

This improvement sets the stage for comparing different LoRA configurations, which we explore next by examining whether higher adapter ranks lead to further gains.

Where LoRA r = 8 clearly outperforms the baseline

The most striking pattern emerges when comparing cases where LoRA r = 8 passes but the baseline fails. Across 129 tasks, the baseline model produces code that looks plausible but fails unit tests for a very consistent reason: function-name and interface mismatch.

In many baseline outputs:

- The function name does not match the one expected by the MBPP tests.
- Additional helper functions, print statements, or example usage code are included.
- The model rewrites the task using a different signature than specified.

For example, when asked to implement sort_matrix, the baseline often defines a function like sort_matrix_by_row_sum or introduces unnecessary dependencies (e.g., NumPy), causing the tests to fail immediately with a NameError. In contrast, the LoRA-tuned model reliably defines the exact function name and signature expected by the tests, even when the internal logic is relatively simple.

This pattern repeats across a wide range of tasks: string manipulation, numeric checks, sorting, counting, and list operations. In many cases, the baseline solution is logically correct in isolation, but unusable in the evaluation setting due to interface violation.

Takeaways

These examples highlight a key benefit of LoRA fine-tuning: it helps the model better align with task-specific constraints, not just general Python syntax. The r = 8 model has clearly learned to: 1. Follow instructions more literally 2. Match function names exactly 3. Avoid extraneous code or explanations

At the same time, the remaining failures suggest that higher-level reasoning errors—rather than formatting or syntax—are now the dominant limitation. This explains why syntax accuracy is already near ceiling, while pass rate still leaves room for improvement.

Comparing Rank-8 to Rank-32

We trained the mistral7b model with two different ranks: 8 and 32. At first glance, the lower-rank LoRA configuration outperforms the higher-rank one on several more tasks than the other. We looked to examine why.

```
import json
from typing import Dict

R8_RESULTS = "artifacts/metrics/mistral7b-code-r8-mbpp_results.json"
R32_RESULTS = "artifacts/metrics/mistral7b-code-r32-mbpp_results.json"

with open(R8_RESULTS, "r", encoding="utf-8") as f:
    r8 = json.load(f)

with open(R32_RESULTS, "r", encoding="utf-8") as f:
    r32 = json.load(f)

print("r=8 summary:", r8["summary"])
print("r=32 summary:", r32["summary"])
```

```
r=8 summary: {'total': 500, 'syntax_ok': 498, 'syntax_rate': 0.996, 'pass': 138, 'pass_rate'
r=32 summary: {'total': 500, 'syntax_ok': 500, 'syntax_rate': 1.0, 'pass': 136, 'pass_rate':
```

```
def build_task_map(results: Dict) -> Dict:
    """
    Maps task_id -> result entry
    """
    return {
        ex["task_id"]: ex
        for ex in results["results"]
        if ex.get("task_id") is not None
    }

r8_tasks = build_task_map(r8)
r32_tasks = build_task_map(r32)

common_task_ids = set(r8_tasks.keys()) & set(r32_tasks.keys())
print("Common tasks:", len(common_task_ids))
```

```
Common tasks: 500
```

```
both_pass = []
only_r8_pass = []
only_r32_pass = []
both_fail = []
```

```python
for task_id in sorted(common_task_ids):
    r8_pass = r8_tasks[task_id]["passed"]
    r32_pass = r32_tasks[task_id]["passed"]

    if r8_pass and r32_pass:
        both_pass.append(task_id)
    elif r8_pass and not r32_pass:
        only_r8_pass.append(task_id)
    elif not r8_pass and r32_pass:
        only_r32_pass.append(task_id)
    else:
        both_fail.append(task_id)

print(f"Both pass: {len(both_pass)}")
print(f"Only r=8 passes: {len(only_r8_pass)}")
print(f"Only r=32 passes: {len(only_r32_pass)}")
print(f"Both fail: {len(both_fail)}")
```

```
Both pass: 123
Only r=8 passes: 15
Only r=32 passes: 13
Both fail: 349
```

```python
def show_32_examples(task_ids, k=5):
    for task_id in task_ids[:k]:
        print(f"\nTask {task_id}")
        print("Instruction:", r8_tasks[task_id]["instruction"])
        print("\n--- r=8 ---")
        print("Passed:", r8_tasks[task_id]["passed"])
        print(r8_tasks[task_id]["generated"])
        print("\n--- r=32 ---")
        print("Passed:", r32_tasks[task_id]["passed"])
        print(r32_tasks[task_id]["generated"])

show_32_examples(only_r32_pass, k=3)
```

```
Task 32
Instruction: Write a python function to find the largest prime factor of a given number.
```

```
Important:
- Define the function exactly as `max_Prime_Factors`.
- Begin your answer with `def max_Prime_Factors(`.
- Output only valid Python code.

--- r=8 ---
Passed: False
def max_Prime_Factors(n):
    # Initialize the largest prime factor
    largest_prime_factor = 0

    # Loop through all numbers from 2 to n
    for i in range(2, n+1):
        # Check if i is a prime number
        if is_prime(i):
            # Check if i is larger than the largest prime factor
            if i > largest_prime_factor:
                # Update the largest prime factor
                largest_prime_factor = i

    # Return the largest prime factor
    return largest_prime_factor

def is_prime(n):
    # Check if n is divisible by any number from 2 to n-1
    for i in range(2, n):
        if n % i == 0:
            return False

    # If n is not divisible by any number, it is a prime number
    return True

--- r=32 ---
Passed: True
def max_Prime_Factors(num):
    # Initialize the largest prime factor
    largest_prime_factor = 0

    # Iterate through all the prime numbers
    for i in range(2, num+1):
        # Check if the number is divisible by the prime number
        if num % i == 0:
            # Update the largest prime factor
```

```
            largest_prime_factor = i
            num = num / i

    # Return the largest prime factor
    return largest_prime_factor
```

```
Task 89
Instruction: Write a function to find the closest smaller number than n.

Important:
- Define the function exactly as `closest_num`.
- Begin your answer with `def closest_num(`.
- Output only valid Python code.

--- r=8 ---
Passed: False
def closest_num(n):
    """
    Find the closest smaller number than n.
    """
    closest = 0
    for i in range(1, n):
        if n - i < closest:
            closest = n - i
    return closest

--- r=32 ---
Passed: True
def closest_num(n):
    """
    Find the closest smaller number than n.
    """
    # Initialize the closest number
    closest = 0
    # Iterate over the numbers from 0 to n
    for i in range(n + 1):
        # Check if the current number is smaller than n
        if i < n:
            # Update the closest number if the current number is closer
            if abs(n - i) < abs(n - closest):
                closest = i
    # Return the closest number
    return closest
```

```
Task 151
Instruction: Write a python function to check whether the given number is co-prime or not.

Important:
- Define the function exactly as `is_coprime`.
- Begin your answer with `def is_coprime(`.
- Output only valid Python code.

--- r=8 ---
Passed: False
def is_coprime(a, b):
    if a == 0 or b == 0:
        return False
    if a == b:
        return True
    if a % b == 0:
        return False
    return is_coprime(b % a, a)

--- r=32 ---
Passed: True
def is_coprime(a, b):
    if a == 0 or b == 0:
        return False
    if a == b:
        return True
    if a > b:
        a, b = b, a
    for i in range(2, min(a, b) + 1):
        if a % i == 0 and b % i == 0:
            return False
    return True
```

When LoRA r = 32 Outperforms r = 8

While LoRA with rank 8 generally performs better under strict unit-test constraints, there are clear cases where LoRA r = 32 succeeds on tasks that r = 8 fails. These examples highlight the benefits of increased representational capacity when tasks require deeper algorithmic reasoning rather than simple pattern matching.

1. Largest Prime Factor (Task 32)

In this task, the r = 8 model attempts to solve the problem by: - Enumerating all primes up to n - Selecting the largest prime encountered

However, this approach is fundamentally flawed because it does not verify whether a prime actually divides the input number. As a result, the function often returns the largest prime n, not the largest prime factor of n.

In contrast, the r = 32 model: - Iteratively divides the input number by its factors - Updates the candidate largest prime factor dynamically

This reflects a more accurate understanding of the underlying number-theoretic concept. The ability to combine factorization with state updates suggests that r = 32 can internalize multi-step reasoning patterns that r = 8 struggles to represent.

2. Closest Smaller Number (Task 89)

The r = 8 model fails due to a subtle logical error: - It initializes closest = 0 and compares differences incorrectly - The update condition never triggers for most valid inputs

This indicates a shallow heuristic rather than a full evaluation of the problem constraints.

By contrast, the r = 32 model: - Explicitly evaluates all candidates less than n - Tracks the closest value using absolute differences

Although slightly verbose, this approach correctly handles edge cases and aligns with the problem specification. The r = 32 model demonstrates greater robustness in translating informal problem descriptions into precise control flow.

3. Coprimality Check (Task 151)

The r = 8 solution uses a recursive structure resembling the Euclidean algorithm but implements it incorrectly:

- It terminates early in cases where numbers are equal
- It fails to correctly identify shared factors beyond simple divisibility

This suggests partial recall of a known algorithm without full correctness.

The r = 32 model, on the other hand:

- Systematically checks all possible common divisors
- Avoids recursion-related pitfalls
- Correctly handles all edge cases specified by the unit tests

Here, the increased rank enables the model to choose correctness over brevity, favoring a reliable—if less elegant—solution.

```python
def show_r8_examples(task_ids, k=5):
    for task_id in task_ids[:k]:
        print("=" * 80)
        print(f"Task ID: {task_id}")
        print("Instruction:", r8_tasks[task_id]["instruction"])

        print("\n--- LoRA r=8 (PASSED) ---")
        print(r8_tasks[task_id]["generated"])

        print("\n--- LoRA r=32 (FAILED) ---")
        print("Errors:", r32_tasks[task_id]["errors"])
        print(r32_tasks[task_id]["generated"])

show_r8_examples(only_r8_pass, k=3)
```

```
================================================================================
Task ID: 21
Instruction: Write a function to find m number of multiples of n.

Important:
- Define the function exactly as `multiples_of_num`.
- Begin your answer with `def multiples_of_num(`.
- Output only valid Python code.

--- LoRA r=8 (PASSED) ---
def multiples_of_num(m, n):
    """
    Find m number of multiples of n
    """
    multiples = []
    for i in range(0, m):
        multiples.append(n * (i + 1))
    return multiples

--- LoRA r=32 (FAILED) ---
Errors: ['Execution error: AssertionError: ']
def multiples_of_num(m, n):
    """
    Find m number of multiples of n.
    """
    multiples = []
    for i in range(m):
```

```
        multiples.append(i * n)
    return multiples
================================================================================
Task ID: 30
Instruction: Write a python function to count all the substrings starting and ending with sa

Important:
- Define the function exactly as `count_Substring_With_Equal_Ends`.
- Begin your answer with `def count_Substring_With_Equal_Ends(`.
- Output only valid Python code.

--- LoRA r=8 (PASSED) ---
def count_Substring_With_Equal_Ends(string):
    count = 0
    for i in range(len(string)):
        for j in range(i+1, len(string)+1):
            if string[i] == string[j-1] and string[i:j] != "":
                count += 1
    return count

--- LoRA r=32 (FAILED) ---
Errors: ['Execution error: AssertionError: ']
def count_Substring_With_Equal_Ends(string):
    count = 0
    for i in range(len(string)):
        for j in range(i+1, len(string)+1):
            if string[i] == string[j-1] and string[i:j] != string[i]:
                count += 1
    return count
================================================================================
Task ID: 94
Instruction: Write a function to extract the index minimum value record from the given tuple

Important:
- Define the function exactly as `index_minimum`.
- Begin your answer with `def index_minimum(`.
- Output only valid Python code.

--- LoRA r=8 (PASSED) ---
def index_minimum(tuples):
    min_val = min(tuples, key=lambda x: x[1])
    return min_val[0]
```

```
--- LoRA r=32 (FAILED) ---
Errors: ["Execution error: TypeError: '<' not supported between instances of 'str' and 'float
def index_minimum(tuples):
    min_value = float('inf')
    min_index = 0
    for i, (value, index) in enumerate(tuples):
        if value < min_value:
            min_value = value
            min_index = index
    return min_index
```

When LoRA r = 8 Outperforms r = 32

Although higher-rank LoRA adapters offer greater expressive capacity, several MBPP tasks reveal situations where LoRA r = 8 produces more reliable solutions than r = 32. These cases tend to involve problems with simple, well-defined logic and strict output expectations, where overcomplication can lead to subtle but consequential errors.

1. Generating Multiples of a Number (Task 21)

The r = 8 solution directly follows the problem specification: - it generates the first m positive multiples of n by multiplying n with integers starting from 1 - Violates the problem's implicit requirement that multiples begin at n - Fails unit tests due to off-by-one logic

This approach aligns cleanly with the expected output and passes all unit tests.

In contrast, the r = 32 model introduces a small but critical deviation by starting multiplication at zero. - Starts multiplication at 1, generating [n, 2n, 3n, …] - Closely follows the problem description - Passes all unit tests with a simple loop

This example illustrates a broader pattern: r = 8 tends to favor minimal, specification-driven logic, while r = 32 sometimes diverges due to unnecessary generalization.

2. Task 30 — Counting Substrings with Equal Start and End Characters

r = 32 Error - Adds an unnecessary constraint to exclude certain substrings - Accidentally filters out valid cases (e.g., single-character substrings) - Introduces logic not required by the task definition

r = 8 Correction - Uses a direct nested-loop approach - Counts all substrings where the first and last characters match - Avoids unnecessary conditions and aligns exactly with expected behavior

3. Task 94 — Extracting the Index of the Minimum Value Record

r = 32 Error - Manually tracks minimum values using incorrect tuple unpacking - Attempts comparisons between incompatible types (strings vs floats) - Results in a runtime TypeError

r = 8 Correction - Leverages Python's built-in min function with a lambda key - Correctly identifies the tuple with the smallest value - Returns the expected index without type conflicts

4. Common Error Patterns Observed in r = 32

- Over-engineering simple tasks
- Introducing additional logic that deviates from benchmark assumptions
- Manually reimplementing functionality better handled by Python primitives
- Higher susceptibility to off-by-one and type-handling errors

```
def error_types(ex):
    if not ex["syntax_ok"]:
        return "syntax"
    if ex["errors"]:
        return "runtime_or_logic"
    return "unknown"

from collections import Counter

r8_error_dist = Counter(error_types(r8_tasks[t]) for t in both_fail)
r32_error_dist = Counter(error_types(r32_tasks[t]) for t in both_fail)

print("r=8 error distribution:", r8_error_dist)
print("r=32 error distribution:", r32_error_dist)
```

```
r=8 error distribution: Counter({'runtime_or_logic': 347, 'syntax': 2})
r=32 error distribution: Counter({'runtime_or_logic': 349})
```

Summary Insight

Taken together, these results suggest that bigger adapters are not always better. For many beginner-to-intermediate programming tasks, a lower-rank LoRA configuration can strike a more effective balance between expressivity and constraint adherence. This finding reinforces a central theme of the project: effective fine-tuning is about alignment and discipline, not just capacity.

More broadly, the project demonstrates that accessible, open-source models—when carefully fine-tuned—can meaningfully close the gap with larger, proprietary code assistants, even under limited compute. While these models are not yet a replacement for professional software

engineers, their growing ability to generate correct, readable, and test-passing code raises important questions about the evolving role of entry-level programming and the future of human–AI collaboration in software development.