How Local LLMs Transform Exam Outcomes: Faster, More Accurate Results, and Higher Grades in Introductory Programming Exams

Appendices

APPENDIX

A. Exam set 1

Task 1: Implement a class called `Robot`, that has a field `batteryLevel` of type `int` that shows the current battery level and a field `maxLevel` of type `int` that shows the maximum capacity of the battery, and `chargingRate` of type `int` that shows the time (number of minutes) it takes for battery level to increase by one unit.

Task 2: Include a constructor to initialize `batteryLevel`, `maxLevel`, and `chargingRate` with the values of the constructor parameters.

Task 3: In class `Robot`, implement a method called `getMaxLevel()` that returns the value of `maxLevel` field.

Task 4: In class `Robot`, implement a method `performTask(int requiredEnergy)` that checks whether the required energy to perform a task is less than or equal to the `batteryLevel`. If it is, then it reduces the `batteryLevel` by the required energy and returns `true`. Otherwise, it returns `false`.

Task 5: In class `Robot`, implement a method called `timeToCharge()` that returns a whole number representing the number of minutes that it takes to charge the battery to maximum capacity.

Task 6: In class `Robot`, implement a method called `charge()` that sets the `batteryLevel` to the maximum capacity.

Task 7: Implement a class called `ServiceRobot`, that is a subclass of `Robot`. This class has a field called `taskInfo` of type `Map<String, Integer>` that contains the name of tasks mapped to their required energy. Implement the constructor of the class `ServiceRobot(int m, int b, int c)`. The `taskInfo` should be initialized as an empty map in the constructor.

Task 8: In class `ServiceRobot`, implement a method `defineTask()` that reads a task name and its required energy (included in two consecutive lines) from user input. If the required energy is negative or greater than the maximum battery capacity, it throws an `IllegalArgumentException` with the message *'The value of required energy is not valid'*. Otherwise, it adds the task name mapped to its required energy in `tasksInfo` map.

Task 9: In class `ServiceRobot`, implement a method `charge()` that prints *'Time to charge is ¡x¿ minutes'* where ¡x¿ is the number of minutes it takes to charge the battery to the maximum capacity and sets the `batteryLevel` to `maxLevel`.

B. Exam set 2

Task 1: Implement a class called `Customer` that has the fields: `preferences` of type `Set<String>` for customer preferences, `advertisements` of type `List<String>` for a list of advertisements received, and `maxAdvs` of type `int` for the maximum number of advertisements a customer can have.

Task 2: Include a constructor to initialize `preferences` and `maxAdvs` with its two parameters, setting `advertisements` as an empty list.

Task 3: Implement a method called `addAdvertisement(String adv)` in the `Customer` class which adds `adv` after the last item in advertisements list if the number of its items remains less than or equal to `maxAdvs` and returns `true`. Otherwise, it just returns `false`.

Task 4: Implement a method `readAdvertisements(int n)` that prints per line the last `n` advertisements in the advertisements list, starting from the end of the list, and deletes them from the list. (Note that it prints and deletes all advertisements if the size is less than or equal to n.)

Task 5: Implement a class called `AdvertisingPlatform` that has the fields `subscribers` of type `List<Customer>` and `forbiddenWords` of type `Set<String>`. The constructor `AdvertisingPlatform(Set<String> forbiddenWords)` initializes the `forbiddenWords` field with the parameter of the constructor and initializes the `subscribers` as an empty list.

Task 6: Implement a method called `addCustomer(Customer cm)` that adds the customer `cm` to the subscribers list if it is not already in the list and prints *'customer is added!'*. Otherwise, it just prints *'customer already exists!'*.

Task 7: In class `AdvertisingPlatform`, implement a method called `checkValidity(String adv)` that checks the validity of advertisement `adv` by checking that the advertisement has at most 20 words and does not have any of the words in the forbidden words. If the advertisement is valid it returns `true`. Otherwise it returns `false`.

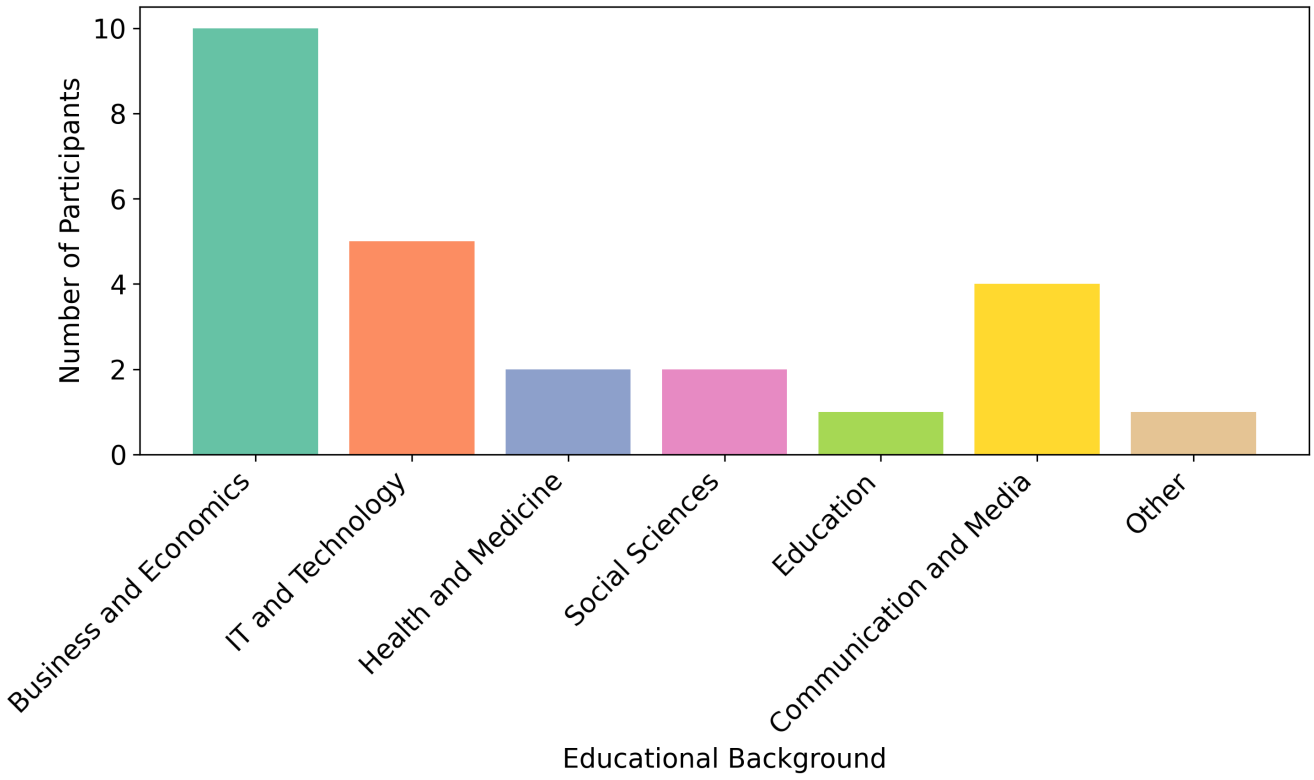

Fig. 1. Distribution of educational background of participants

Each task is awarded points depending on the complexity of the task. For each task, points are awarded if the participant's solution meets the requirements correctly. If the solution is partially correct, the participant is awarded partial points based on a subjective assessment. Points are awarded for the following:

- Correct method signature: Award points based on expected return values and expected parameters.
- Correct syntax and logic: Award points for correct syntax and implementation of accurate logical solutions.
- Efficient and optimal solutions: Award points for solutions that demonstrate simplicity and effectiveness.
- Proper naming conventions and code formatting: Award points for good code readability and organization.

C. Exam set 1 (120 points):

- Task 1: 15 points (Defining a class with multiple fields)
- Task 2: 10 points (Implementing a constructor)
- Task 3: 5 points (Simple getter method)
- Task 4: 15 points (Implementing logic with conditional statements)
- Task 5: 15 points (Implementing logic with mathematical operations)
- Task 6: 10 points (Updating object state)
- Task 7: 15 points (Inheritance and constructor implementation)
- Task 8: 20 points (User input handling, exception handling, and map manipulation)
- Task 9: 15 points (String formatting and updating object state)

D. Exam set 2 (120 points):

- Task 1: 20 points (Defining a class with multiple fields of different data types)
- Task 2: 10 points (Implementing a constructor with initialization)
- Task 3: 15 points (List manipulation with conditional logic)
- Task 4: 20 points (Printing and modifying a list based on user input)
- Task 5: 15 points (Defining a class with multiple fields and constructor implementation)
- Task 6: 15 points (List manipulation with conditional logic)
- Task 7: 25 points (String manipulation, conditional logic, and word counting)

Since the total points available for both exam sets are 120, we can map the point ranges to the Danish grade scale as follows:

- 12 (Excellent): 108 - 120 points (90% - 100%)
- 10 (Very Good): 90 - 107.9 points (75% - 89.9%)
- 7 (Good): 72 - 107.9 points (60% - 74.9%)
- 4 (Fair): 48 - 71.9 points (40% - 59.9%)
- 02 (Adequate): 36 - 47.9 points (30% - 39.9%)
- 00 (Inadequate): 24 - 35.9 points (20% - 29.9%)
- -3 (Unacceptable): Below 24 points (Below 20%)

E. Student's solution to Task 7 & 8 from exam set 1

```
    import java.util.Map;

public class ServiceRobot extends Robot {
    Map<String,Integer> taskInfo;

    public ServiceRobot(int m, int b, int c){
        super(b, m, c);
        this.taskInfo = new Map<>();
    }
    public void defineTask(String taskName, int requiredEnergy){
        if (requiredEnergy < 0 || requiredEnergy < this.batteryLevel){
            // throw IllegalArgumentException here
            throw new Exception("The value of required energy is not valid");
        }
        this.taskInfo.put(taskName, requiredEnergy);
    }
}
```

Task 7: Implementing the `ServiceRobot` Class Constructor

Points available: 15

- **Correct method signature (5 points)**
  - **Expected:** `public ServiceRobot(int m, int b, int c)`
  - **Provided:** `public ServiceRobot(int m, int b, int c)`
  - **Points:** 5/5
- **Correct syntax and logic (8 points)**
  - The student initializes the superclass with `super(b, m, c);`
  - The student initializes `taskInfo` as `new Map<>();`, which is incorrect. The correct initialization should be `new HashMap<>();`
  - **Points:** 7/8 (1 points deducted for incorrect initialization)
- **Code readability and organization (2 points)**
  - The code is reasonably readable, but using `Map<>` without proper type initialization can be confusing.
  - **Points:** 1/2

Total for Task 7: 13/15

Task 8: Implementing the `defineTask` Method

Points available: 20

- **Correct method signature (5 points)**
  - **Expected:** `public void defineTask(String taskName, int requiredEnergy)`
  - **Provided:** `public void defineTask(String taskName, int requiredEnergy)`
  - **Points:** 5/5
- **Correct syntax and logic (10 points)**
  - The method checks if `requiredEnergy` is less than 0 or less than `batteryLevel` instead of checking if it's greater than `batteryLevel`.
  - The method throws a generic `Exception` instead of `IllegalArgumentException`.
  - **Points:** 5/10 (3 points deducted for incorrect condition, 2 point deducted for wrong exception type)
- **Error handling (3 points)**
  - The method does handle errors by throwing an exception but uses the wrong type.
  - **Points:** 1/3
- **Map manipulation (2 points)**
  - The method correctly adds the task to the `taskInfo` map.
  - **Points:** 2/2

Total for Task 8: 13/20

14	1	0	0.012s
tests	failures	ignored	duration

92%
successful

Failed tests **Tests**

Test	Duration	Result
testAddAdvertisementWhenListIsFull	0s	passed
testAddAdvertisementWhenListIsNotFull	0.002s	passed
testAddAdvertisementWithEmptyList	0s	passed
testAddCustomerAlreadyExists	0s	passed
testAddCustomerSuccess	0s	passed
testAdvertisingPlatformConstructor	0s	passed
testCheckValidityForbiddenWord	0.004s	failed
testCheckValidityTooLong	0s	passed
testCheckValidityValid	0.003s	passed
testCustomerConstructorTask2	0s	passed
testCustomerFieldsTask1	0.001s	passed
testReadAdvertisementsWhenNIsEqualToSize	0.001s	passed
testReadAdvertisementsWhenNIsGreaterThanSize	0.001s	passed
testReadAdvertisementsWhenNIsLessThanSize	0s	passed

Fig. 2.

Class Definition

```
public class ClassName {
    // class body
}
```

Method Definition

```
public returnType methodName(parameters) {
    // method body
}
```

Constructor

```
public ClassName(parameters) {
    // constructor body
}
```

Main Method

```
public static void main(String[] args) {
    // entry point of program
}
```

Declaring Variables

```
type variableName = value;
```

Primitive Data Types

```
int, double, float, boolean, char
```

Object Data Types

```
String, Array, Set, List, Map
```

If-Else

```
if (condition) {
    // if block
} else {
    // else block
}
```

For Loop

```
for (initialization; condition; update) {
    // code block
}
```

While Loop

```
while (condition) {
    // code block
}
```

List (ArrayList)

```
List<Type> listName = new ArrayList<>();
```

Set (HashSet)

```
Set<Type> setName = new HashSet<>();
```

Map (HashMap)

```
Map<KeyType, ValueType> mapName = new HashMap<>();
```

Throwing Exceptions

```
throw new ExceptionType("Error message");
```

Extending a Class

```
public class SubClass extends SuperClass {
    // subclass body
}
```

Arrays

```
Type[] arrayName = new Type[size];
```

Access Modifiers

```
private, public, protected
```

Reading from Console

```
Scanner scanner = new Scanner(System.in);
String input = scanner.nextLine();
int number = scanner.nextInt();
```

Printing to Console

```
System.out.println("Message");
```

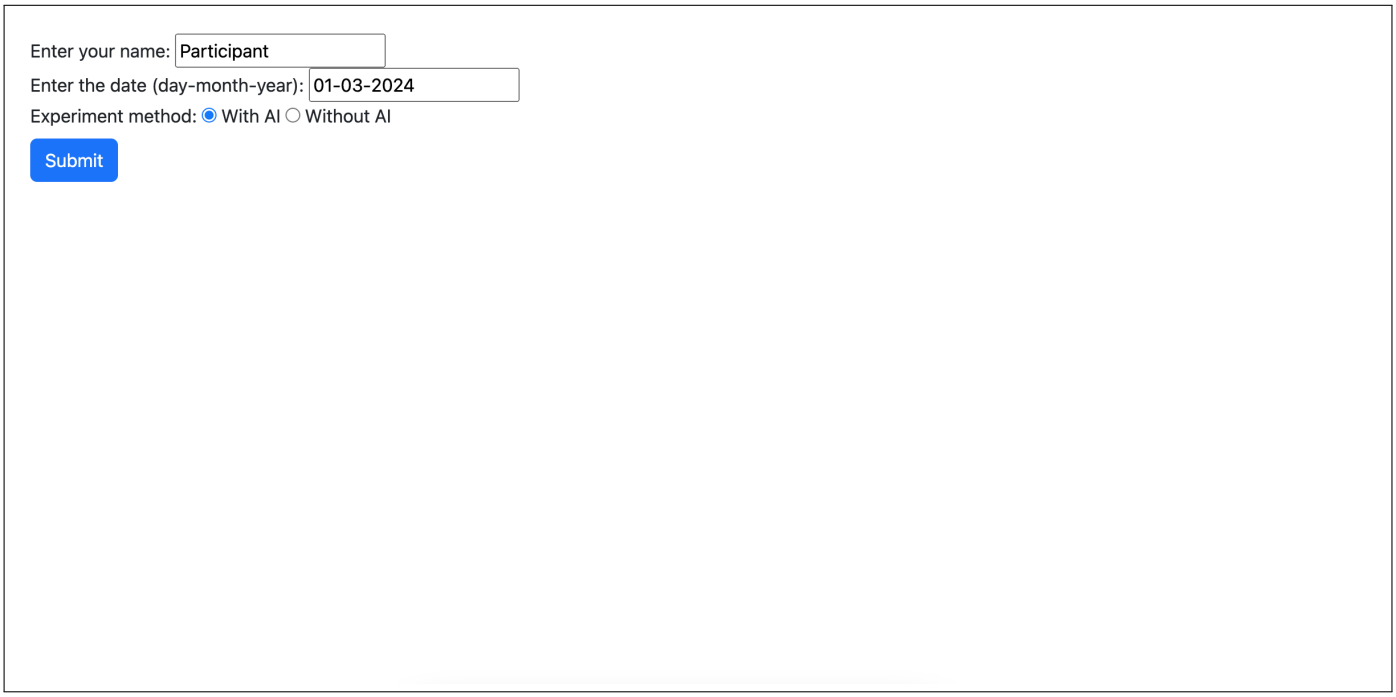

Fig. 3. Page 1 - Set data and initialize with or without AI

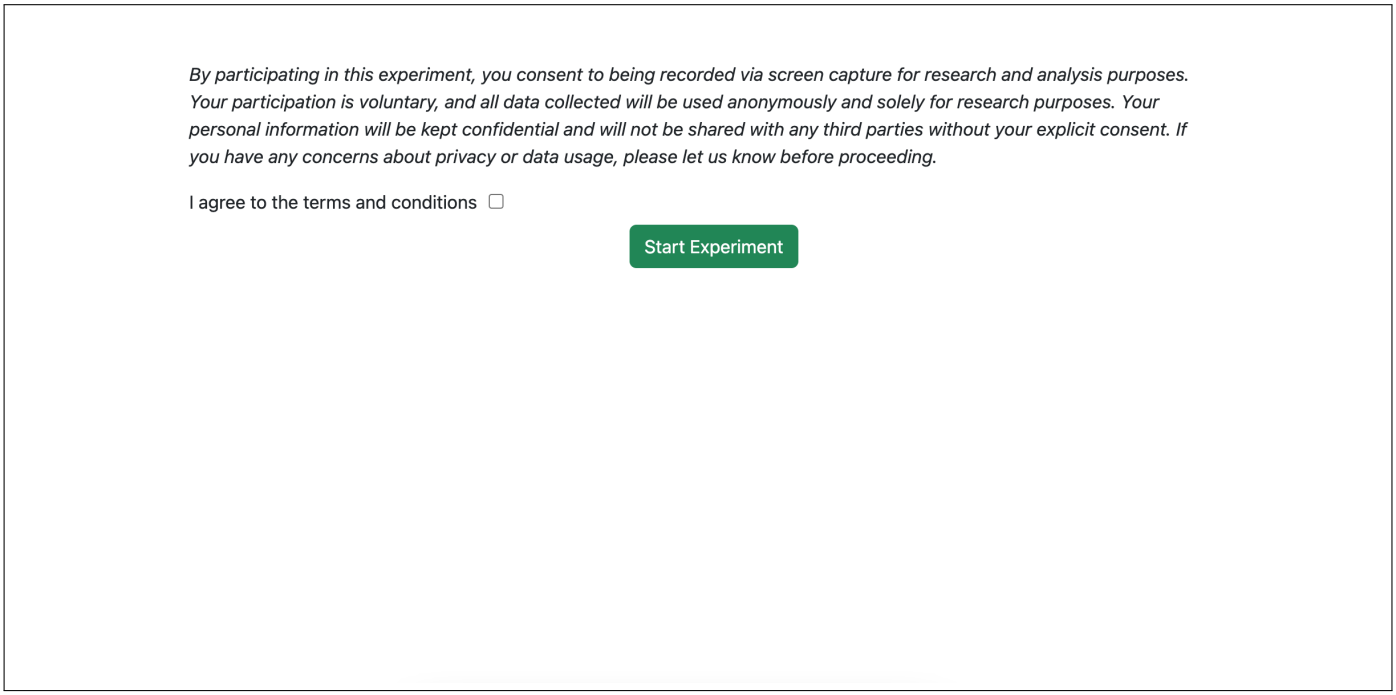

Fig. 4. Page 2 - Terms and conditions

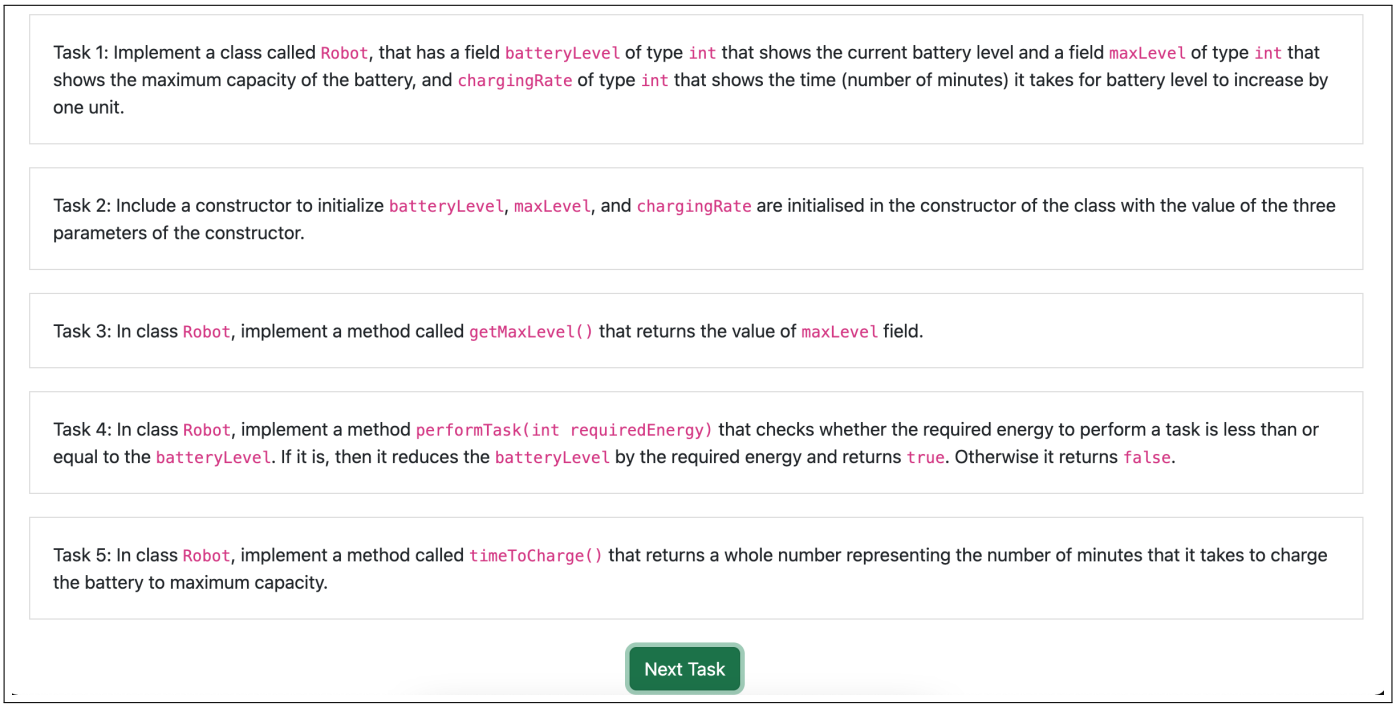

Fig. 5. Page 3 - Example of tasks (Participant have loaded 5 tasks)

Software Experimentation Pre-Questionnaire
A questionnaire to collect basic data before the experiment.

1) In what subject was your bachelor?
2) How many years of developer experience do you have? *Required
3) How many years of experience do you have with Java? *Required
4) How many months is it since you coded in Java? *Required
5) How many years of experience do you have with Visual Studio Code? *Required
6) How much experience in years do you have in using an LLM (ChatGPT, CoPilot, Gemini)? *Required
7) Which operating system do you usually use? *Required
    - Windows
    - MacOS
    - Linux
    - Something else