

# LLM path finder experiment

Made by: Bertalan Lichter (MBSEKD)

## Abstract

---

This is a homework assignment for the Software Technology course (fall 2023). The aim of the assignment is to test the ability of LLMs when it comes to writing relatively simple code and their ability to test these classes themselves.

## My unit test

---

```
package imsc.softtech;

import org.junit.jupiter.api.*;

import static org.junit.jupiter.api.Assertions.*;

class PathfinderTest {
    private Pathfinder pathFinder;

    @BeforeEach
    void beforeEach() {
        pathFinder = new DijkstraPathFinder2();
    }

    @Test
    void singleEdgeTest() {
        pathFinder.addEdge(0, 1, 1);
        assertEquals(1, pathFinder.getShortestPathLength(0, 1));
    }

    @Test
    void noPathTest () {
        pathFinder.addEdge(0, 1, 1);
        pathFinder.addEdge(2, 3, 1);
        assertEquals(-1, pathFinder.getShortestPathLength(0, 3));
    }

    @Test
    void emptyGraphTest () {
        assertEquals(-1, pathFinder.getShortestPathLength(0, 1));
    }

    @Test
    void weightedDiamond () {
        pathFinder.addEdge(0, 1, 1);
        pathFinder.addEdge(1, 2, 1);
        pathFinder.addEdge(0, 3, 1);
        pathFinder.addEdge(2, 3, 0);
        assertEquals(1, pathFinder.getShortestPathLength(0, 3));
    }
}
```

```

    @Test
    void
    testThatChainedZeroWeightedEdgesArePreferredOverOneWeightedDirectEdge () {
        pathFinder.addEdge(0, 1, 0);
        pathFinder.addEdge(1, 2, 0);
        pathFinder.addEdge(2, 3, 0);
        pathFinder.addEdge(3, 4, 0);
        pathFinder.addEdge(0, 4, 1);
        assertEquals(0, pathFinder.getShortestPathLength(0, 4));
    }

    @Test
    void testThatDefiningMultipleEdgesDoesThrow () {
        pathFinder.addEdge(0, 1, 1);
        assertThrows(IllegalStateException.class, () ->
        pathFinder.addEdge(0, 1, 1));
    }

    @Test
    void testThatOnlyValidEdgesWithValidWeightsAreAccepted () {
        assertDoesNotThrow(() -> pathFinder.addEdge(0, 1, 0));
        assertDoesNotThrow(() -> pathFinder.addEdge(0, 2, 1));
        assertThrows(IllegalArgumentException.class, () ->
        pathFinder.addEdge(0, 3, -1));
        assertThrows(IllegalArgumentException.class, () ->
        pathFinder.addEdge(0, 4, -2));
        assertThrows(IllegalArgumentException.class, () ->
        pathFinder.addEdge(0, 5, 2));
    }

    @Test
    void testDirectionality () {
        pathFinder.addEdge(0, 1, 1);
        pathFinder.addEdge(2, 1, 1);
        assertEquals(-1, pathFinder.getShortestPathLength(0, 2));
    }

    @Test
    void testThatWeightIsZeroWhenStartAndEndNodesAreTheSame () {
        pathFinder.addEdge(0, 1, 1);
        pathFinder.addEdge(1, 0, 1);
        assertEquals(0, pathFinder.getShortestPathLength(0, 0));
    }

    @Test
    void testThatItHandlesLoops () {
        pathFinder.addEdge(0, 0, 1);
        pathFinder.addEdge(0, 1, 1);
        pathFinder.addEdge(1, 2, 1);
        pathFinder.addEdge(2, 2, 1);
        assertEquals(2, pathFinder.getShortestPathLength(0, 2));
    }

    @Test

```

```

void complexGraph () {

    /*
    * Graph: (~transit graph)
    *
    * 0 = 1 = 2 = 3 = 4
    *   |0|       /\   \/
    *   10       5 = 6
    *   ||       \
    *   LL= 20 = 30
    *           |0|
    *           300 = 400 = 500
    * */

    pathFinder.addEdge(0, 1, 1);
    pathFinder.addEdge(1, 0, 1);

    pathFinder.addEdge(1, 2, 1);
    pathFinder.addEdge(2, 1, 1);

    pathFinder.addEdge(2, 3, 1);
    pathFinder.addEdge(3, 2, 1);

    pathFinder.addEdge(3, 4, 1);
    pathFinder.addEdge(5, 3, 1);

    pathFinder.addEdge(4, 6, 1);
    pathFinder.addEdge(6, 5, 1);

    pathFinder.addEdge(1, 10, 0);
    pathFinder.addEdge(10, 1, 0);

    pathFinder.addEdge(10, 20, 1);
    pathFinder.addEdge(20, 10, 1);

    pathFinder.addEdge(20, 30, 1);
    pathFinder.addEdge(30, 20, 1);

    pathFinder.addEdge(30, 5, 1);
    pathFinder.addEdge(5, 30, 1);

    pathFinder.addEdge(30, 300, 0);
    pathFinder.addEdge(300, 30, 0);

    pathFinder.addEdge(300, 400, 1);
    pathFinder.addEdge(400, 300, 1);

    pathFinder.addEdge(400, 500, 1);
    pathFinder.addEdge(500, 400, 1);

    assertEquals(1, pathFinder.getShortestPathLength(0, 10));
    assertEquals(4, pathFinder.getShortestPathLength(0, 4));
    assertEquals(3, pathFinder.getShortestPathLength(400, 3));
    assertEquals(5, pathFinder.getShortestPathLength(3, 400));
    assertEquals(3, pathFinder.getShortestPathLength(3, 5));

```

```

        assertEquals(1, pathFinder.getShortestPathLength(5, 3));
        assertEquals(0, pathFinder.getShortestPathLength(30, 300));
        assertEquals(5, pathFinder.getShortestPathLength(500, 4));
    }
}

```

## Solution 1

---

### Setup

Model: GPT 3.5

Link to the chat: <https://chat.openai.com/share/f2b07954-5bf1-44a1-91c1-5ccb37856359>

### Prompt

Hi! You are in a coding interview. Your task is to create a pathfinding implementation in java. The interface has already been written for you:

```

/** * Find the shortest path in a weighted directed graph. */ interface
PathFinder { /** * Adds an edge to the graph. * * @param from the ID of the
node at the start of the edge. * @param to the ID of the node at the end of
the edge. * @param weight the weight of the edge, either 1 or 0. * @throws
IllegalArgumentException if the edge weight is invalid. * @throws
IllegalStateException if the edge was already present in the graph. */ void
addEdge(int from, int to, int weight); /** * Calculates the weight of the
shortest path in the graph. * * @param source the ID of the source node of
the path. * @param target the ID of the target node of the path. * @returns
the weight of the shortest path or -1 if there is no such path. */ int
getShortestPathLength(int source, int target); }

```

Your job is to create an implementation of this interface!

### The solution of the LLM

*Note: I have changed the name of the class, as all of them had the same name and added the package declaration on the top!*

```

package imsc.softtech;

import java.util.*;

public class DijkstraPathFinder1 implements PathFinder {
    private final Map<Integer, Map<Integer, Integer>> graph;

    public DijkstraPathFinder1() {
        this.graph = new HashMap<>();
    }

    @Override
    public void addEdge(int from, int to, int weight) {
        if (weight != 0 && weight != 1) {
            throw new IllegalArgumentException("Edge weight must be either
0 or 1.");
        }

        graph.computeIfAbsent(from, k -> new HashMap<>());
        if (graph.get(from).containsKey(to)) {

```

```

        throw new IllegalStateException("Edge already present in the
graph.");
    }

    graph.get(from).put(to, weight);
}

@Override
public int getShortestPathLength(int source, int target) {
    Map<Integer, Integer> distance = new HashMap<>();
    PriorityQueue<Node> priorityQueue = new
PriorityQueue<>(Comparator.comparingInt(distance::get));

    distance.put(source, 0);
    priorityQueue.add(new Node(source, 0));

    while (!priorityQueue.isEmpty()) {
        int current = priorityQueue.poll().vertex;

        if (current == target) {
            return distance.get(current);
        }

        if (graph.containsKey(current)) {
            for (int neighbor : graph.get(current).keySet()) {
                int newDistance = distance.get(current) +
graph.get(current).get(neighbor);

                if (!distance.containsKey(neighbor) || newDistance <
distance.get(neighbor)) {
                    distance.put(neighbor, newDistance);
                    priorityQueue.add(new Node(neighbor, newDistance));
                }
            }
        }

        return -1; // No path found
    }

    private static class Node {
        private final int vertex;
        private final int distance;

        public Node(int vertex, int distance) {
            this.vertex = vertex;
            this.distance = distance;
        }
    }
}

```

## Solution 2

---

### Setup

Model: GPT 3.5

Link to the chat: <https://chat.openai.com/share/82f6e79a-c18d-430a-8094-811eba01b364>

### Prompt

Hi! I am writing my homework for a university assignment. My task is to have LLMs generate java implementations of a pre-written interface and then run unit tests on them (first mine and then tests written by the respective LLM). You are (one of several) such chosen LLM. You have to write a path finder implementation for the following java interface:

```
/**
 * Find the shortest path in a weighted directed graph.
 */
interface Pathfinder {
    /**
     * Adds an edge to the graph.
     *
     * @param from the ID of the node at the start of the edge.
     * @param to the ID of the node at the end of the edge.
     * @param weight the weight of the edge, either 1 or 0.
     * @throws IllegalArgumentException if the edge weight is invalid.
     * @throws IllegalStateException if the edge was already present in the
graph.
     */
    void addEdge(int from, int to, int weight);

    /**
     * Calculates the weight of the shortest path in the graph.
     *
     * @param source the ID of the source node of the path.
     * @param target the ID of the target node of the path.
     * @returns the weight of the shortest path or -1 if there is no such
path.
     */
    int getShortestPathLength(int source, int target);
}
```

### The solution of the LLM

*Note: I have changed the name of the class, as all of them had the same name and added the package declaration on the top!*

```
package imsc.softtech;

import java.util.*;

public class DijkstraPathFinder2 implements Pathfinder {
    private Map<Integer, Map<Integer, Integer>> graph;

    public DijkstraPathFinder2() {
        this.graph = new HashMap<>();
    }
}
```

```

    }

    @Override
    public void addEdge(int from, int to, int weight) {
        if (weight != 0 && weight != 1) {
            throw new IllegalArgumentException("Invalid edge weight. It
should be either 0 or 1.");
        }

        graph.computeIfAbsent(from, k -> new HashMap<>()).put(to, weight);
    }

    @Override
    public int getShortestPathLength(int source, int target) {
        Map<Integer, Integer> distances = new HashMap<>();
        PriorityQueue<Node> minHeap = new
PriorityQueue<>(Comparator.comparingInt(distances::get));
        Set<Integer> visited = new HashSet<>();

        minHeap.add(new Node(source, 0));
        distances.put(source, 0);

        while (!minHeap.isEmpty()) {
            Node currentNode = minHeap.poll();
            int current = currentNode.id;

            if (current == target) {
                return distances.get(current);
            }

            if (visited.contains(current)) {
                continue;
            }

            visited.add(current);

            for (Map.Entry<Integer, Integer> neighborEntry :
graph.getDefault(current, Collections.emptyMap()).entrySet()) {
                int neighbor = neighborEntry.getKey();
                int newDistance = distances.get(current) +
neighborEntry.getValue();

                if (!distances.containsKey(neighbor) || newDistance <
distances.get(neighbor)) {
                    distances.put(neighbor, newDistance);
                    minHeap.add(new Node(neighbor, newDistance));
                }
            }
        }

        return -1; // No path found
    }

    private static class Node {
        private final int id;
    }

```

```

        private final int distance;

        private Node(int id, int distance) {
            this.id = id;
            this.distance = distance;
        }
    }
}

```

## Solution 3

---

### Setup

Model: GPT 3.5

Link to the chat: <https://chat.openai.com/share/6689d473-388a-416e-a94d-37622cd220f4>

### Prompt

Hi! Your task will be to implement Dijkstra's algorithm in java. The implementation will have to match an interface given by me. The graph built will be directed and the edges will have either a weight of 1 or 0. You will only have to return the weight of the best route, not the sequence of nodes. Each node will only be represented by an (integer) index. The interface, you have to implement:

```

/**
 * Find the shortest path in a weighted directed graph.
 */
interface Pathfinder {
    /**
     * Adds an edge to the graph.
     *
     * @param from the ID of the node at the start of the edge.
     * @param to the ID of the node at the end of the edge.
     * @param weight the weight of the edge, either 1 or 0.
     * @throws IllegalArgumentException if the edge weight is invalid.
     * @throws IllegalStateException if the edge was already present in the
    graph.
    */
    void addEdge(int from, int to, int weight);

    /**
     * Calculates the weight of the shortest path in the graph.
     *
     * @param source the ID of the source node of the path.
     * @param target the ID of the target node of the path.
     * @returns the weight of the shortest path or -1 if there is no such
    path.
    */
    int getShortestPathLength(int source, int target);
}

```



## The solution of the LLM

*Note: I have changed the name of the class, as all of them had the same name and added the package declaration on the top!*

```
package imsc.softtech;

import java.util.*;

class DijkstraPathFinder3 implements Pathfinder {

    private final Map<Integer, Map<Integer, Integer>> graph;

    public DijkstraPathFinder3() {
        this.graph = new HashMap<>();
    }

    @Override
    public void addEdge(int from, int to, int weight) {
        if (weight != 0 && weight != 1) {
            throw new IllegalArgumentException("Edge weight must be either
0 or 1");
        }

        graph.computeIfAbsent(from, k -> new HashMap<>()).put(to, weight);
    }

    @Override
    public int getShortestPathLength(int source, int target) {
        Map<Integer, Integer> distances = new HashMap<>();
        PriorityQueue<int[]> minHeap = new
PriorityQueue<>(Comparator.comparingInt(a -> a[1]));

        minHeap.add(new int[]{source, 0});
        distances.put(source, 0);

        while (!minHeap.isEmpty()) {
            int[] current = minHeap.poll();
            int currentNode = current[0];
            int currentDistance = current[1];

            if (currentNode == target) {
                return currentDistance;
            }

            if (!graph.containsKey(currentNode)) {
                continue;
            }

            for (Map.Entry<Integer, Integer> neighbor :
graph.get(currentNode).entrySet()) {
                int neighborNode = neighbor.getKey();
                int newDistance = currentDistance + neighbor.getValue();

                if (!distances.containsKey(neighborNode) || newDistance <
distances.get(neighborNode)) {
```

```

        distances.put(neighborNode, newDistance);
        minHeap.add(new int[]{neighborNode, newDistance});
    }
}

return -1; // No path found
}
}

```

## Initial thoughts and analysis of the code

First of all, the code compiled with no errors. There were also very few warnings. Of the warnings that the IDE (IntelliJ IDEA with Sonarlint plugin for the warnings) did show, some were just indications that some fields may be final or that a field could be converted to a local variable. However, one warning was genuine: in the first two implementations the internal class called Node had a field called distance that was never used, it instead calculated it on the fly via a HashMap. This was the mark of an LLM: during training it saw several examples of an internal class, called Node and it always had a distance field, but the model then did not correctly associate it with the right implementations.

As to the safety of the code, it was perfectly safe. The `getShortestPathLength` functions were pure, and the `addEdge` functions' side effects were only limited to the updating of the `graph` field (in a correct way that would not cause data-loss).

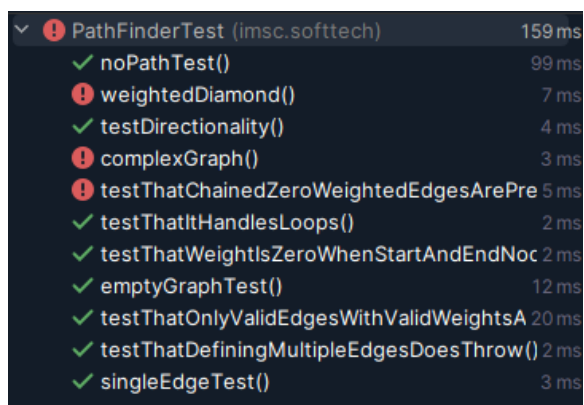
## Test results

To run the tests and see the coverage, I ran them using IntelliJ IDEA's test runner with coverage.

### Tests made by me

#### Results

	Successful tests / all tests	Line coverage	Branch coverage
Implementation 1	8/11	100%	94%
Implementation 2	7/11	96%	87%
Implementation 3	10/11	100%	100%



PathFinderTest (imsc.softtech)		159 ms
✓	noPathTest()	99 ms
✗	weightedDiamond()	7 ms
✓	testDirectionality()	4 ms
✗	complexGraph()	3 ms
✗	testThatChainedZeroWeightedEdgesArePre	5 ms
✓	testThatItHandlesLoops()	2 ms
✓	testThatWeightsIsZeroWhenStartAndEndNoc	2 ms
✓	emptyGraphTest()	12 ms
✓	testThatOnlyValidEdgesWithValidWeightsA	20 ms
✓	testThatDefiningMultipleEdgesDoesThrow()	2 ms
✓	singleEdgeTest()	3 ms

#### Screenshots

PathFinderTest (imsc.softtech)	143 ms
noPathTest()	83 ms
weightedDiamond()	6 ms
testDirectionality()	3 ms
complexGraph()	3 ms
testThatChainedZeroWeightedEdgesArePre	2 ms
testThatItHandlesLoops()	3 ms
testThatWeightsZeroWhenStartAndEndNoc	3 ms
emptyGraphTest()	2 ms
testThatOnlyValidEdgesWithValidWeightsA	17 ms
testThatDefiningMultipleEdgesDoesThrow()	7 ms
singleEdgeTest()	14 ms

PathFinderTest (imsc.softtech)	154 ms
noPathTest()	103 ms
weightedDiamond()	4 ms
testDirectionality()	3 ms
complexGraph()	3 ms
testThatChainedZeroWeightedEdgesArePre	3 ms
testThatItHandlesLoops()	3 ms
testThatWeightsZeroWhenStartAndEndNoc	3 ms
emptyGraphTest()	3 ms
testThatOnlyValidEdgesWithValidWeightsA	11 ms
testThatDefiningMultipleEdgesDoesThrow	16 ms
singleEdgeTest()	2 ms

Element ^	Class, %	Method, %	Line, %	Branch, %
imsc.softtech	33% (2/6)	35% (5/14)	33% (29/87)	34% (17/50)
DijkstraPathFinder1	100% (2/2)	100% (5/5)	100% (29/29)	94% (17/18)
DijkstraPathFinder2	0% (0/2)	0% (0/5)	0% (0/31)	0% (0/16)
DijkstraPathFinder3	0% (0/1)	0% (0/3)	0% (0/26)	0% (0/16)
Main	0% (0/1)	0% (0/1)	0% (0/1)	100% (0/0)
PathFinder	100% (0/0)	100% (0/0)	100% (0/0)	100% (0/0)
imsc.softtech	33% (2/6)	35% (5/14)	34% (30/87)	28% (14/50)
imsc.softtech	16% (1/6)	21% (3/14)	29% (26/87)	32% (16/50)
DijkstraPathFinder1	0% (0/2)	0% (0/5)	0% (0/29)	0% (0/18)
DijkstraPathFinder2	0% (0/2)	0% (0/5)	0% (0/31)	0% (0/16)
DijkstraPathFinder3	100% (1/1)	100% (3/3)	100% (26/26)	100% (16/16)
Main	0% (0/1)	0% (0/1)	0% (0/1)	100% (0/0)
PathFinder	100% (0/0)	100% (0/0)	100% (0/0)	100% (0/0)

## Tests generated by the LLM

### Setup

Model: GPT 3.5

Link to the chat: <https://chat.openai.com/share/56edec2e-d3f2-4078-a285-92ea0ceb83af>

### Prompt

Hi! Your task is to generate JUnit5 unit tests for the given java interface:

```

/**
 * Find the shortest path in a weighted directed graph.
 */
interface Pathfinder {
    /**
     * Adds an edge to the graph.
     *
     * @param from the ID of the node at the start of the edge.
     * @param to the ID of the node at the end of the edge.
     * @param weight the weight of the edge, either 1 or 0.
     * @throws IllegalArgumentException if the edge weight is invalid.
     * @throws IllegalStateException if the edge was already present in the
graph.
     */
    void addEdge(int from, int to, int weight);

    /**
     * Calculates the weight of the shortest path in the graph.
     *
     * @param source the ID of the source node of the path.
     * @param target the ID of the target node of the path.
     * @returns the weight of the shortest path or -1 if there is no such
path.
     */
    int getShortestPathLength(int source, int target);
}

```

Source code by the LLM (commented)

```

package imsc.softtech;

import org.junit.jupiter.api.BeforeEach;
import org.junit.jupiter.api.Test;
import static org.junit.jupiter.api.Assertions.*;

class GeneratedTest {

    private Pathfinder pathFinder;

    @BeforeEach
    void setUp() {
        // Initialize the Pathfinder implementation before each test
        pathFinder = new DijkstraPathFinder1(); // Replace with the actual
implementation class
    }

    /**
     * EDITOR: This checks that we can add an edge (addEdge happy path)
     */
    @Test
    void addEdge_ValidInput_AddsEdgeSuccessfully() {
        pathFinder.addEdge(1, 2, 1);
        // Add assertions to verify that the edge was added successfully
        // You might want to use additional methods to check the state of
the graph
    }
}

```

```

    }

    /**
     * EDITOR: This checks that we do not accept invalid weights
     */
    @Test
    void addEdge_InvalidWeight_ThrowsIllegalArgumentException() {
        // Add assertions to check that IllegalArgumentException is thrown
        for invalid weight

        // EDITOR: this is a good test value, but I would add some more
        assertThrows(IllegalArgumentException.class, () ->
pathFinder.addEdge(1, 2, 2));
    }

    /**
     * EDITOR: This checks that duplicate edges are not allowed
     */
    @Test
    void addEdge_DuplicateEdge_ThrowsIllegalStateException() {
        // Add assertions to check that IllegalStateException is thrown for
a duplicate edge
        pathFinder.addEdge(1, 2, 1);
        // EDITOR: this is a good assertion
        assertThrows(IllegalStateException.class, () ->
pathFinder.addEdge(1, 2, 1));
    }

    /**
     * EDITOR: This is a pathfinding happy path
     */
    @Test
    void getShortestPathLength_ExistingPath_ReturnsCorrectLength() {
        // Add edges to the graph to create a known path
        pathFinder.addEdge(1, 2, 1);
        pathFinder.addEdge(2, 3, 1);
        int shortestPathLength = pathFinder.getShortestPathLength(1, 3);
        // EDITOR: correct assertion
        assertEquals(2, shortestPathLength);
    }

    /**
     * EDITOR: This checks for the case of no path existing between the
source and the target
     */
    @Test
    void getShortestPathLength_NoPath_ReturnsMinusOne() {
        // Add assertions to check that -1 is returned when there is no
path
        int shortestPathLength = pathFinder.getShortestPathLength(1, 4);
        // EDITOR: correct assertion
        assertEquals(-1, shortestPathLength);
    }
}

```

## Test results

	Successful tests / all tests	Line coverage	Branch coverage
Implementation 1	5/5	100%	100%
Implementation 2	4/5	100%	96%
Implementation 3	4/5	100%	96%

## Thoughts on the generated tests

The LLM wrote pretty good tests. The main problem, I had with them is that there were not enough of them. It checked for some obvious edge-cases and some happy paths, but the I found that the happy paths were too simple, and the edge-cases were too few in number. For example, it did not check for the directedness of the graph. Nor did it check that the length of the best path is actually the minimal path (there were no test cases with 2 or more paths connecting the source and the target). Based on these results, it would not be a great idea to let the LLMs test their own code, because a poorly written code can look perfectly functional when tested with poorly written tests.

## Fixing the code

---

Here, I just told the LLM the name of the test and the output (the failed assertion or the error). I thought that this would closely simulate someone with little coding experience or with little knowledge of the used language and framework.

### Implementation 1

I could not get the LLM to correct its mistake after several tries and some additional context. The messages can be found [here](#).

### Implementation 2

I had no luck here either. The messages can be found [here](#).

### Implementation 3

There was only one minor mistake in this implementation: it didn't check whether the new edge had existed or not upon adding it. This was a relatively easy fix that it managed to do at the second try. The messages can be found [here](#).

It modified the `addEdge` function to look like this:

```
@Override
    public void addEdge(int from, int to, int weight) {
        if (weight != 0 && weight != 1) {
            throw new IllegalArgumentException("Edge weight must be either
0 or 1");
        }

        if (graph.computeIfAbsent(from, k -> new HashMap<>()).put(to,
weight) != null) {
            throw new IllegalStateException("Edge already present in the
graph");
        }
    }
```

## Conclusion

---

Writing code with chat-GPT is a long way from being perfect. However, it has come a long way and it can produce good results if you help it understand the problem and throw in the right keywords that it associates with the good implementations for the given use case.

## Sources

---

### The chats

1. <https://chat.openai.com/share/f2b07954-5bf1-44a1-91c1-5ccb37856359>
2. <https://chat.openai.com/share/82f6e79a-c18d-430a-8094-811eba01b364>
3. <https://chat.openai.com/share/6689d473-388a-416e-a94d-37622cd220f4>

### GitHub repository

[https://github.com/lichterberci/sofftech\\_imsc](https://github.com/lichterberci/sofftech_imsc)