

Vibe Coding a Car Racing Simulator (Failed)

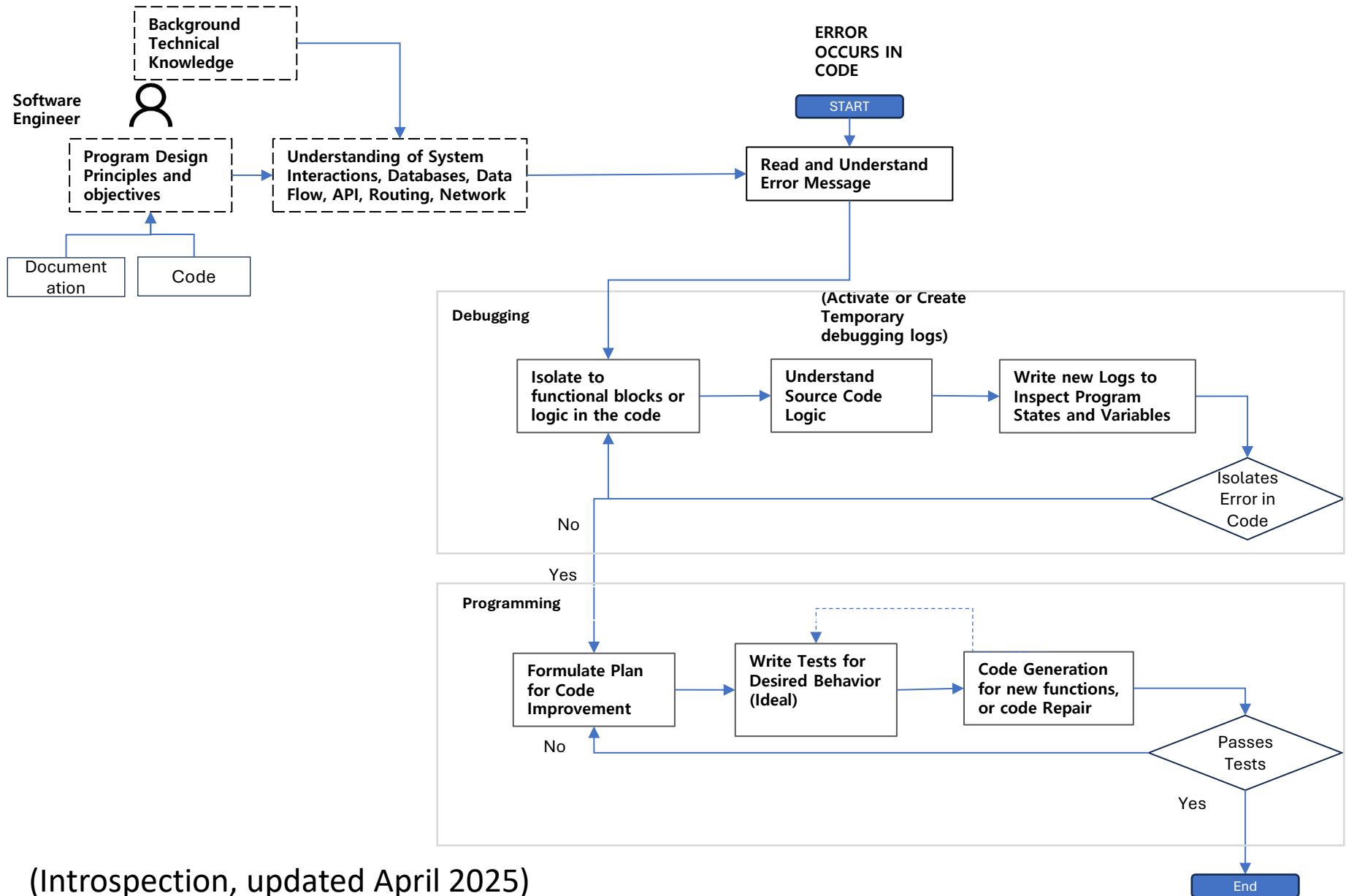
Suzanna Sia

Nov 2024

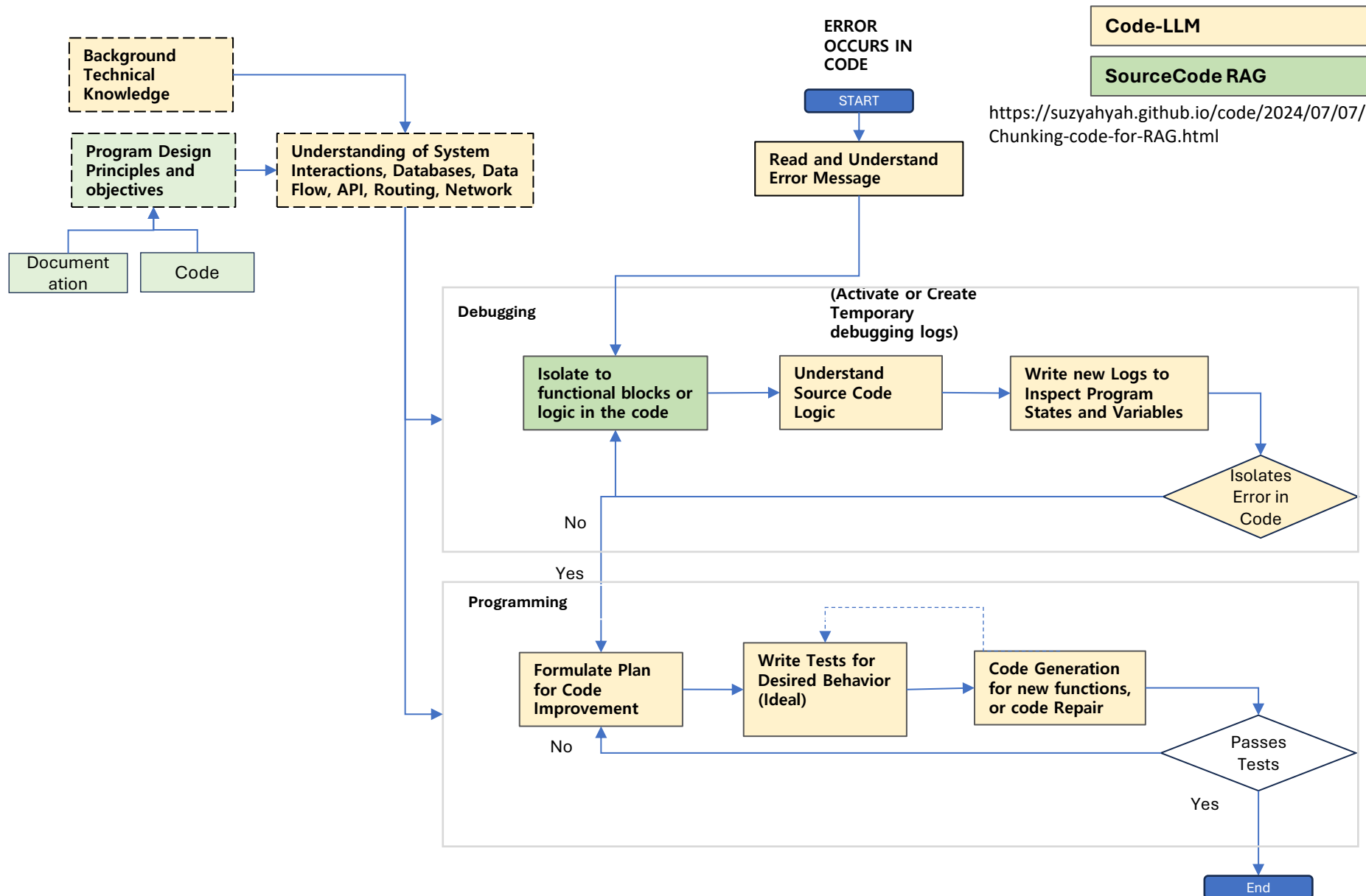
The Original Presentation is here:

https://suzyahyah.github.io/assets/LimitationsOfSourceCodeAgents_SSIA.pdf

Troubleshooting and Code Generation workflow without LLMs



Troubleshooting and Code Generation workflow with LLMs



Summary

1. Steering LLMs towards Code Generation depends on the Programmer providing the relevant context to the Model, for understanding the Program's Design Principles, System Understanding and Domain Understanding.

However if we can provide ALL context to the LLM, then we should be able to replace programmers?

No, because debugging and generating new code in physical control systems is extremely difficult. It's not fix patterns that you can rehash like in constructing certain types of games or UI.

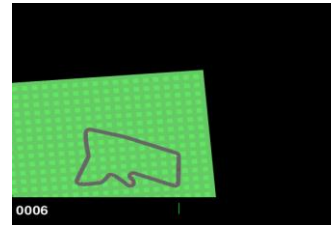
Chat GPT Programmer Workflow

Gen Z
Chat-GPT
Programmer

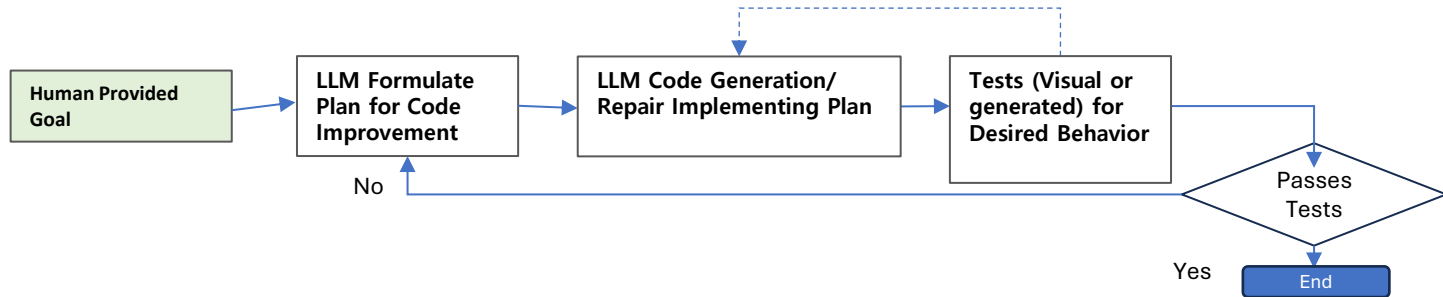


Start

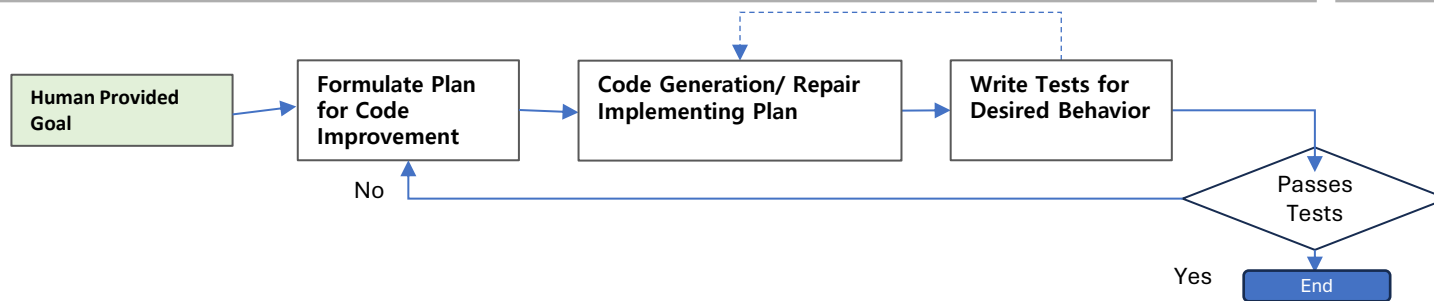
The Car is going off-track
and not making the turn
early enough



Top-down Image of
the Car-racing Track



Formulating Plan for Code Improvement



Step 0: (Human provides Goal)

“Generate the high level logic for a program which will navigate a car on a track.

The track is surrounded by grass and there are no other obstacles.

The only goal is to stay on the track and reach the destination.

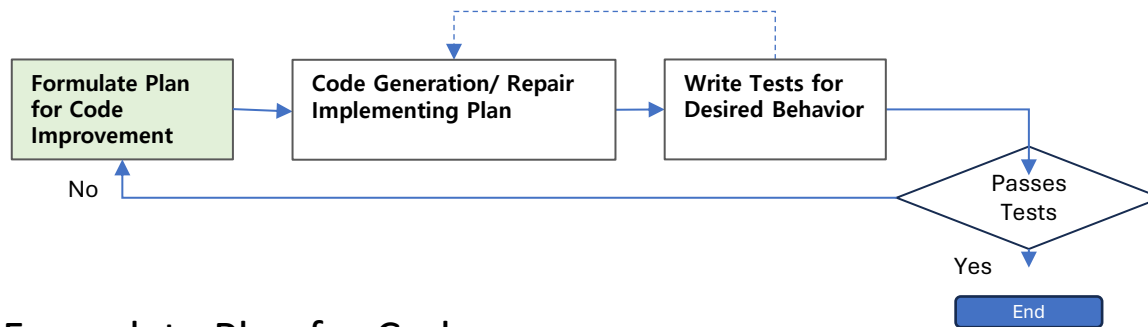
There are four possible actions, steer left, steer right, go forward, brake.

The only thing the program can observe is the pixel image of a top-down view of where the car is. Which might or might not be on the track. “

Summary

1. Steering LLMs towards Code Generation depends on the Programmer providing the relevant context to the Model, for understanding the Program's Design Principles, System Understanding and Domain Understanding.
2. Debugging and generating new code in physical control systems is extremely difficult. It's not fix patterns that you can rehash like in constructing certain types of games or UI.
3. "Formulating Plan for Code Improvement" can generally works well at a high (and not very useful level).

Formulating Plan for Code Improvement

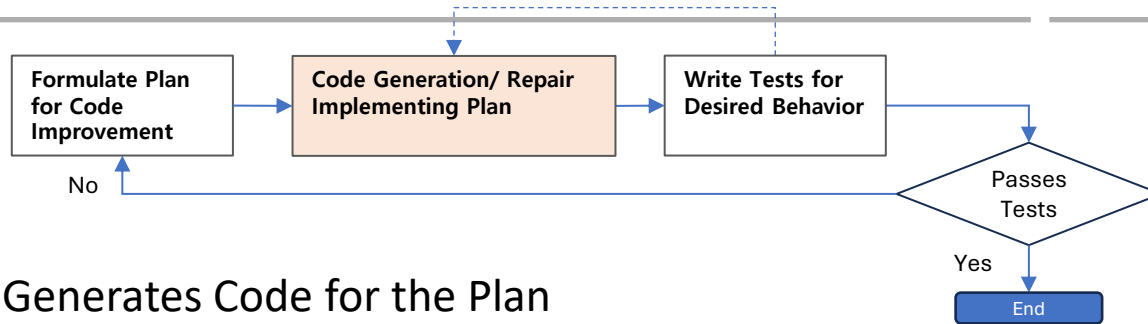


Step 1: Agent Formulate Plan for Code

Main Control Loop

- While the car has not reached the destination:
 1. **Determine Position Relative to Track**
 - Assess the car's location:
 - **In Track:** If the car is on the track based on pixel analysis.
 - **Off Track:** If the car is off the track (i.e., on the grass).
 2. **Decision Making**
 - Based on the car's relative position:
 - **If On Track:**
 - Check for upcoming turns:
 - If a turn is detected, decide whether to:
 - **Brake:** If speed needs to be reduced.
 - **Steer Left/Right:** Adjust the steering to follow the track.
 - If no turns are imminent:
 - **Action:** Go forward.
 - **If Off Track:**
 - Determine the direction to steer based on pixel analysis:
 - If the car is off to the left, **steer right** to realign.
 - If the car is off to the right, **steer left** to realign.
 3. **Execute Action**
 - Perform the chosen action (steer left, steer right, go forward, brake).
 - Update the car's state (position and orientation) based on the action.

Formulating Plan for Code Improvement

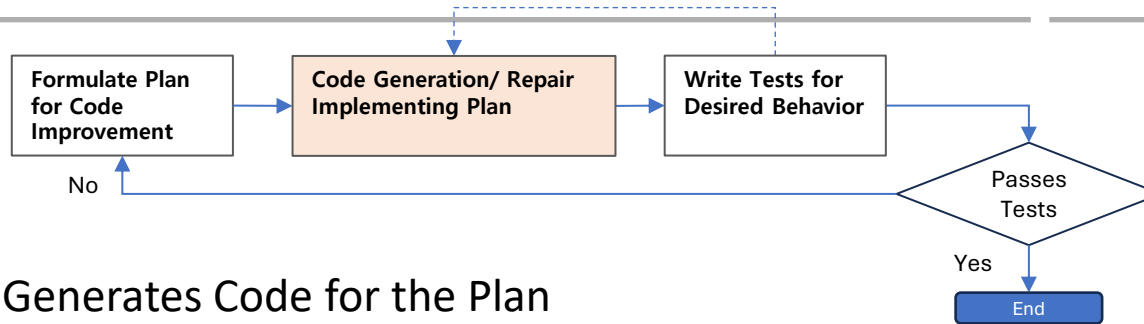


Step 2: Agent Generates Code for the Plan



Oops

Formulating Plan for Code Improvement



Step 2: Agent Generates Code for the Plan

Potential Errors

Wrong High-level Plan

Wrong Implementation of Logic

Wrong assumptions over available inputs

Wrong hyperparameters

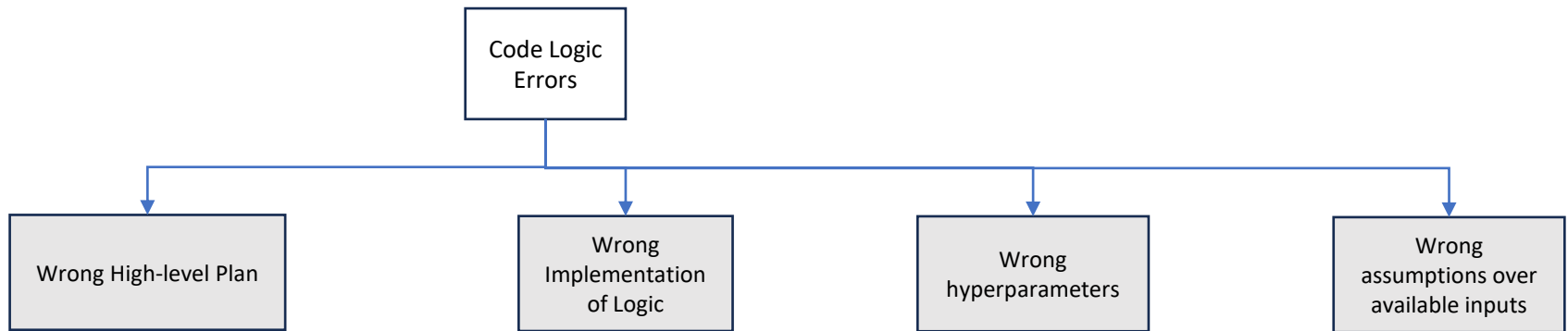
```
# Check if car is within the bounds of the track mask
if 0 <= x < track_width and 0 <= y < track_height:
    if track_mask[y][x] == 1: # Assuming 1 represents track pixels
        on_track = True
    else:
        on_track = False
else:
    on_track = False

# Decision making based on the car's position
if on_track:
    # Check for upcoming turns (simplified logic)
    if detect_turn(track_mask, car_position):
        # Determine if braking is necessary
        if speed > desired_speed: # Assuming desired_speed is defined elsewhere
            action = "brake"
        else:
            # Determine direction to steer based on upcoming turn
            action = determine_steering_direction(track_mask, car_position)
    else:
        action = "go forward"
```



Oops

Step 2: Agent Generates Code for the Plan: Code Logic Errors



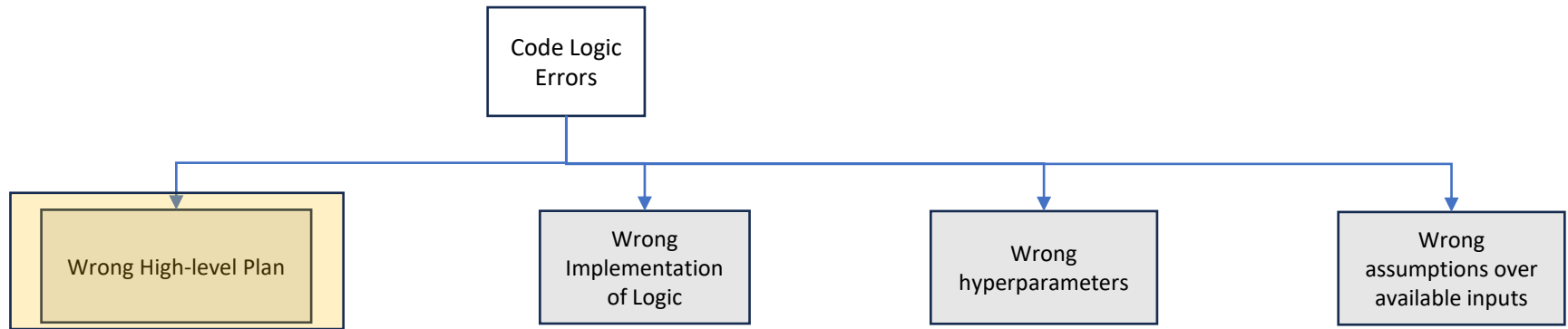
```
# Check if the car is on the track
if track_mask[int(car_y), int(car_x)] == 0:
    capture_screenshot(image, f"images_debug/off-track.step{steps}.png")
```



Is this line correctly written?

How can the model verify this line is as intended?

Step 2: Agent Generates Code for the Plan: Code Logic Errors



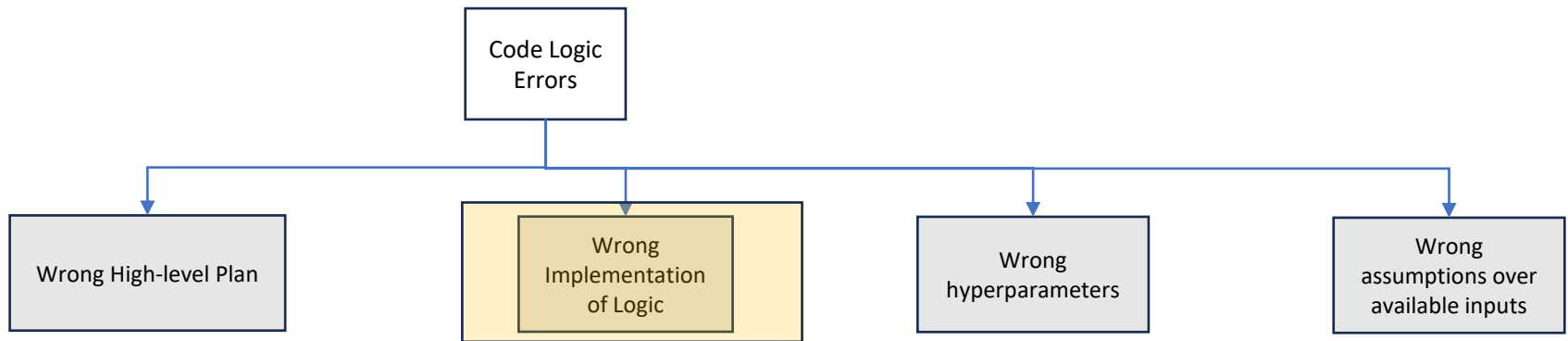
```
# Check if the car is on the track
if track_mask[int(car_y), int(car_x)] == 0:
    capture_screenshot(image, f"images_debug/off-track.step{steps}.png")
```



Is this line correctly written?

How can the model verify this line is as intended?

Step 2: Agent Generates Code for the Plan: Code Logic Errors



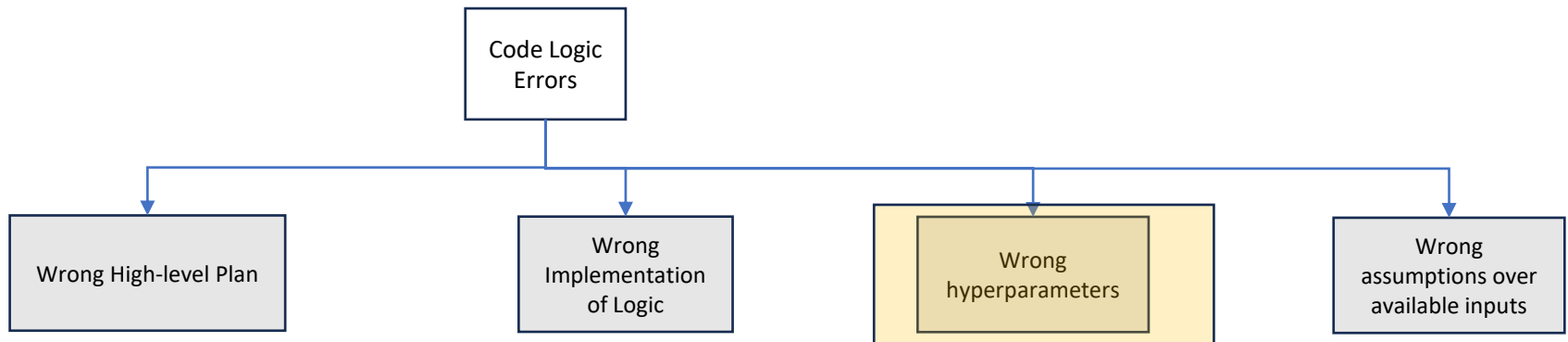
```
# Check if the car is on the track
if track_mask[int(car_y), int(car_x)] == 0:
    capture_screenshot(image, f"images_debug/off-track.step{steps}.png")
```



Is this line correctly written?

How can the model verify this line is as intended?

Step 2: Agent Generates Code for the Plan: Code Logic Errors



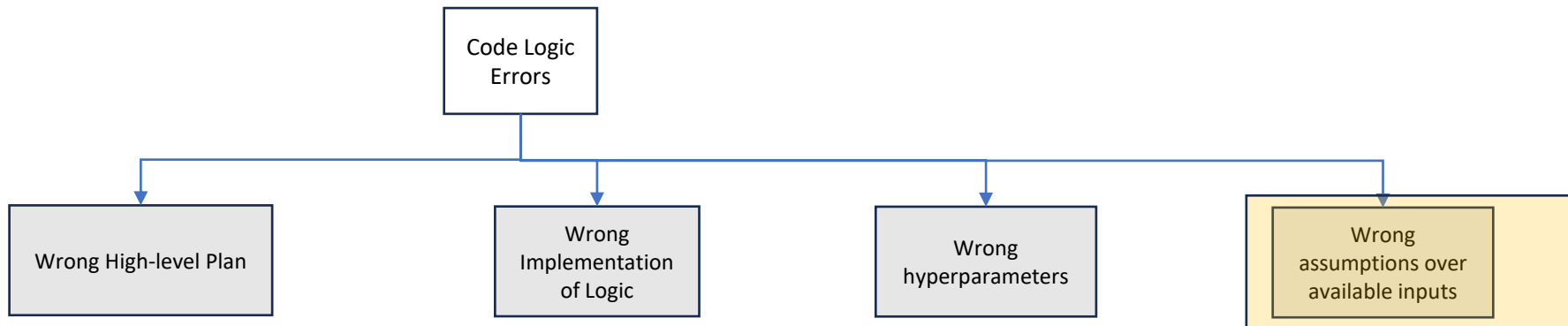
```
# Check if the car is on the track
if track_mask[int(car_y), int(car_x)] == 0:
    capture_screenshot(image, f"images_debug/off-track.step{steps}.png")
```



Is this line correctly written?

How can the model verify this line is as intended?

Step 2: Agent Generates Code for the Plan: Code Logic Errors



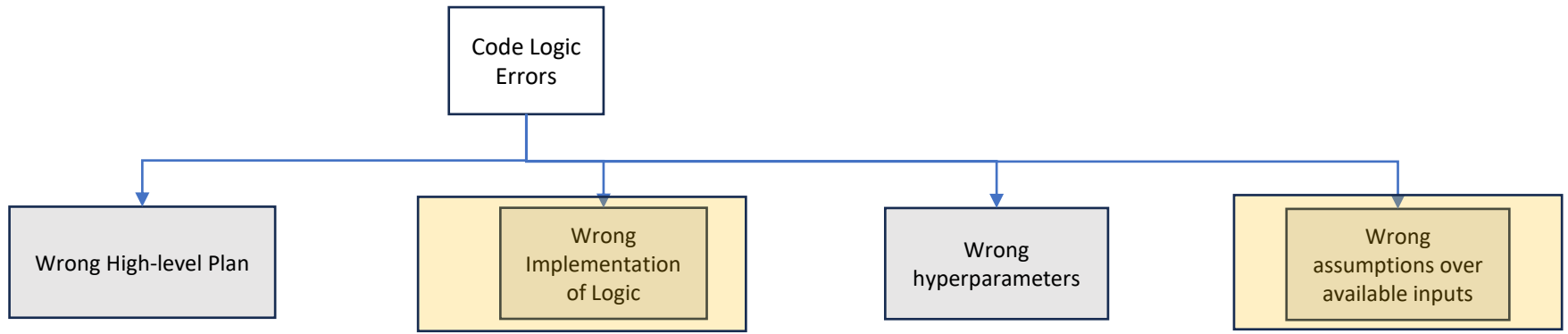
```
# Check if the car is on the track
if track_mask[int(car_y), int(car_x)] == 0:
    capture_screenshot(image, f"images_debug/off-track.step{steps}.png")
```



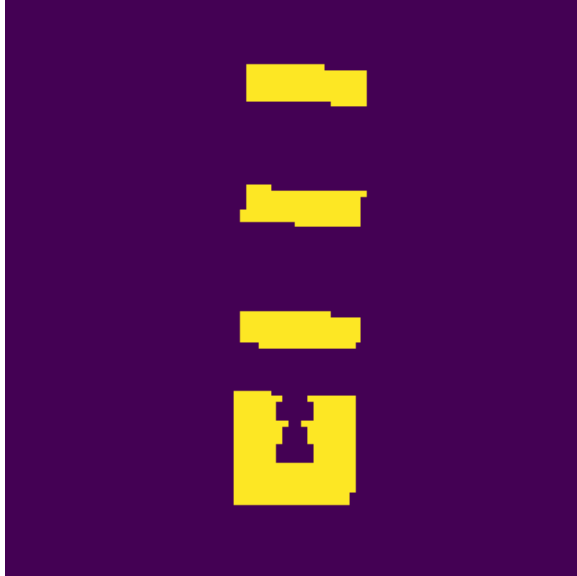
Is this line correctly written?

How can the model verify this line is as intended?

Step 2: Agent Generates Code for the Plan: Code Logic Errors



Using the track mask to detect if the car is on or off-track was a bad idea.

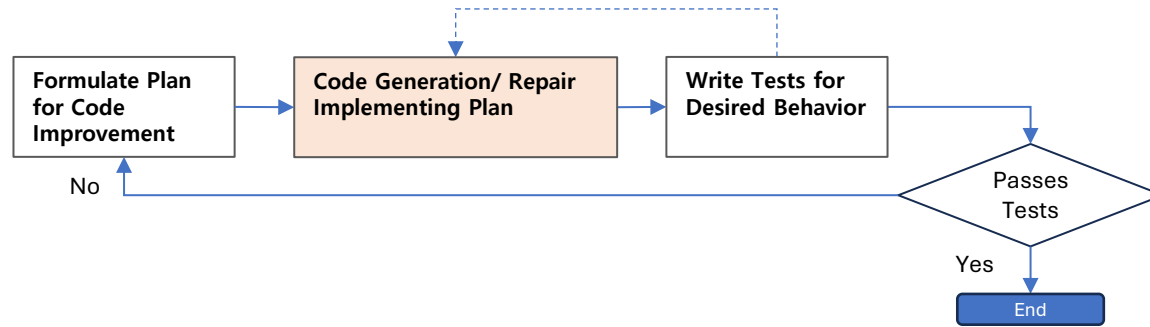


"Track Mask" which the Program uses, but the LLM cannot "see".

Summary

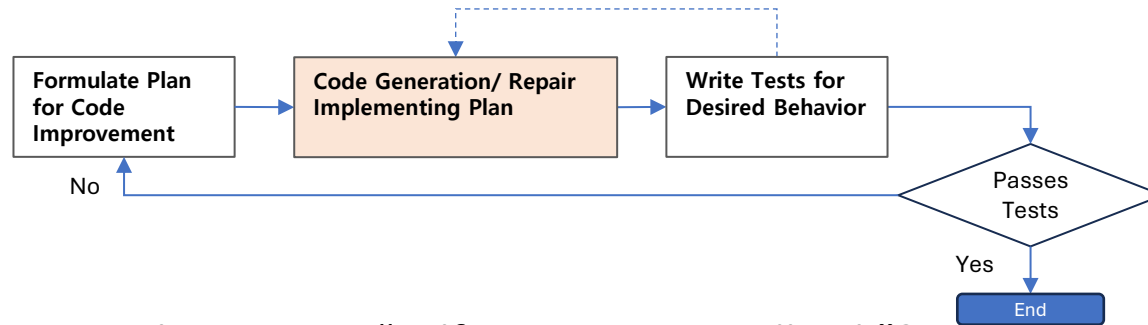
1. Steering LLMs towards Code Generation depends on the Programmer providing the relevant context to the Model, for understanding the Program's Design Principles, System Understanding and Domain Understanding.
2. Debugging and generating new code in physical control systems is extremely difficult. It's not fix patterns that you can rehash like in constructing certain types of games or UI.
3. "Formulating Plan for Code Improvement" can generally works well at a high (and not very useful level).
4. Logic errors can be due to wrong high level plan, wrong implementation logic, wrong hyperparameters, or wrong input assumptions. Neither the code agent or Human has vision over what is happening.

Step 2: Agent Generates Code for the Plan

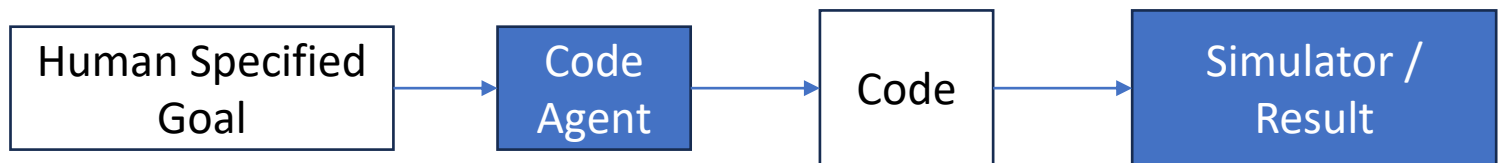


How about using “Self-Corrective Feedback”?

Limitations of Agentic-LLM corrective feedback



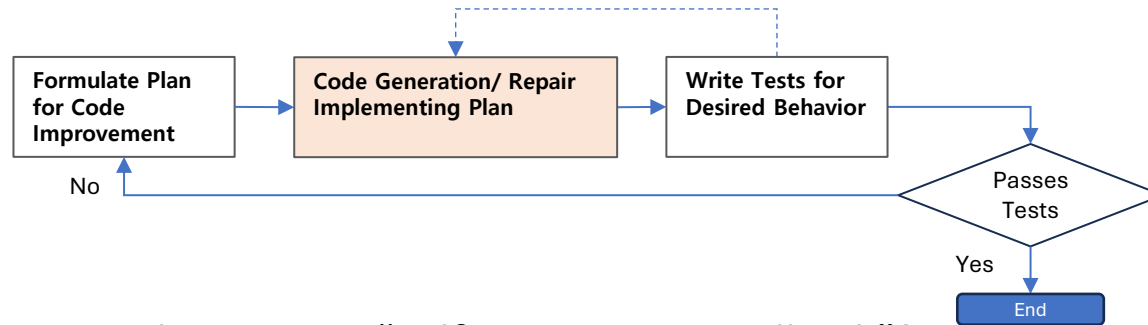
How about using “Self-Corrective Feedback”?



Code Agent sees

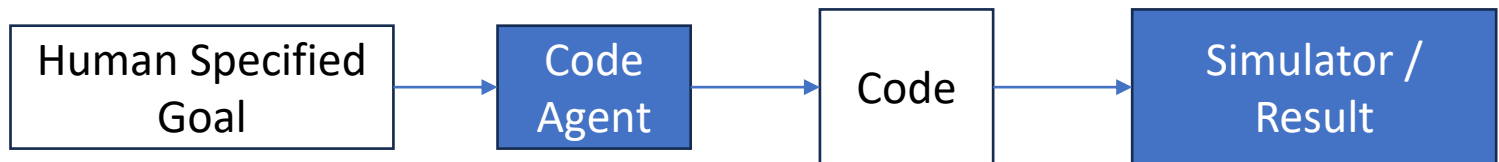
(1) Human Specified Goals

Limitations of Agentic-LLM corrective feedback



How about using “Self-Corrective Feedback”?

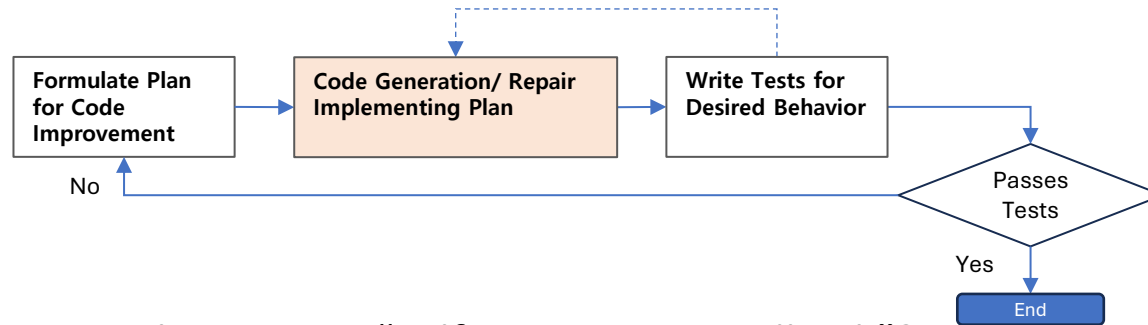
(1) Easy: Execution Error



Code Agent sees

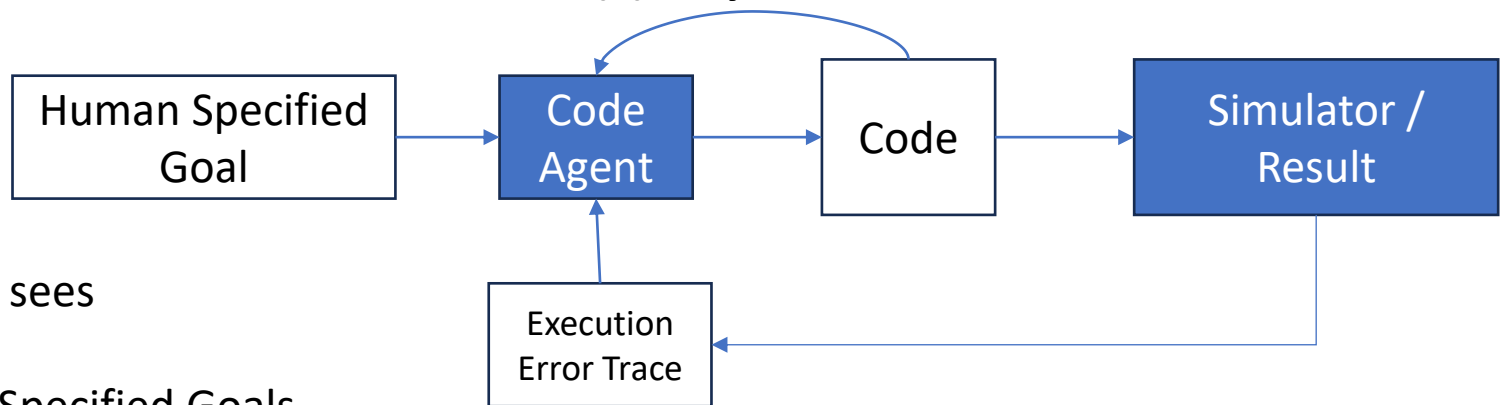
(1) Human Specified Goals

Limitations of Agentic-LLM corrective feedback



How about using “Self-Corrective Feedback”?

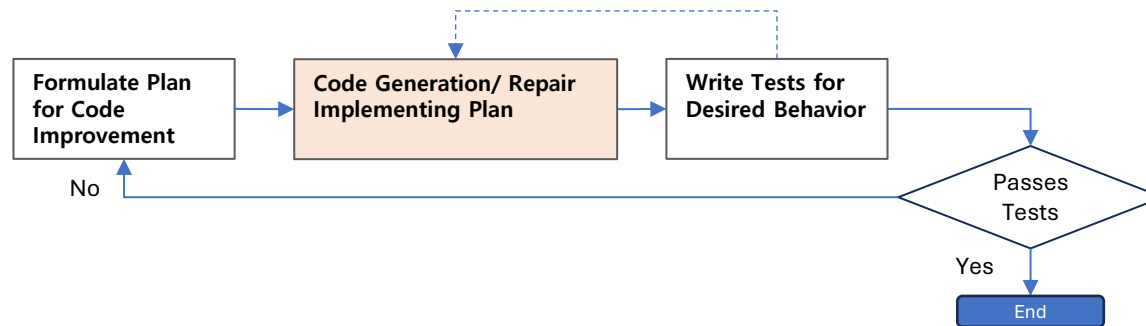
(1) Easy: Execution Error



Code Agent sees

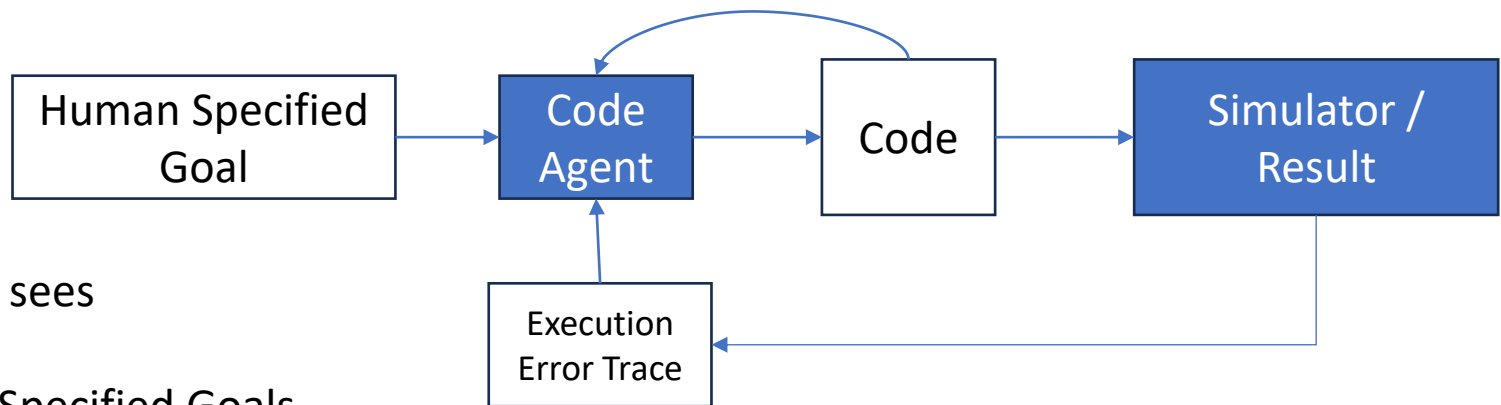
- (1) Human Specified Goals,
- (2) Execution error Trace
- (3) Current Code

Limitations of Agentic-LLM corrective feedback



How about using “Self-Corrective Feedback”?

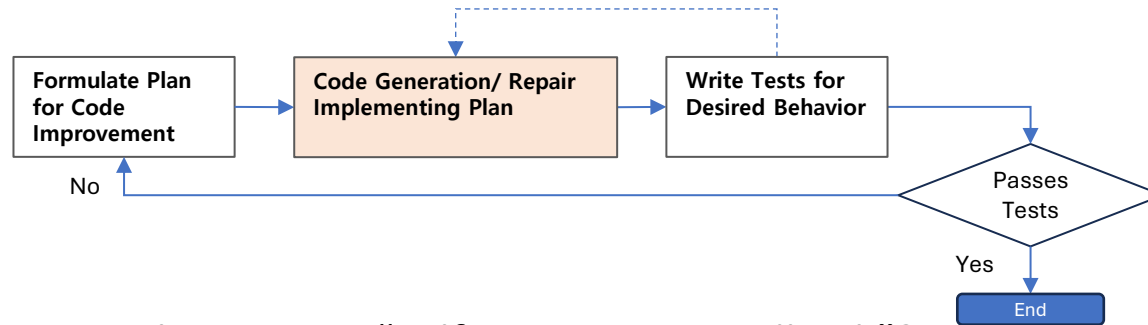
(2) Medium Stuck: No more explicit execution errors, but cannot reach human specified goal



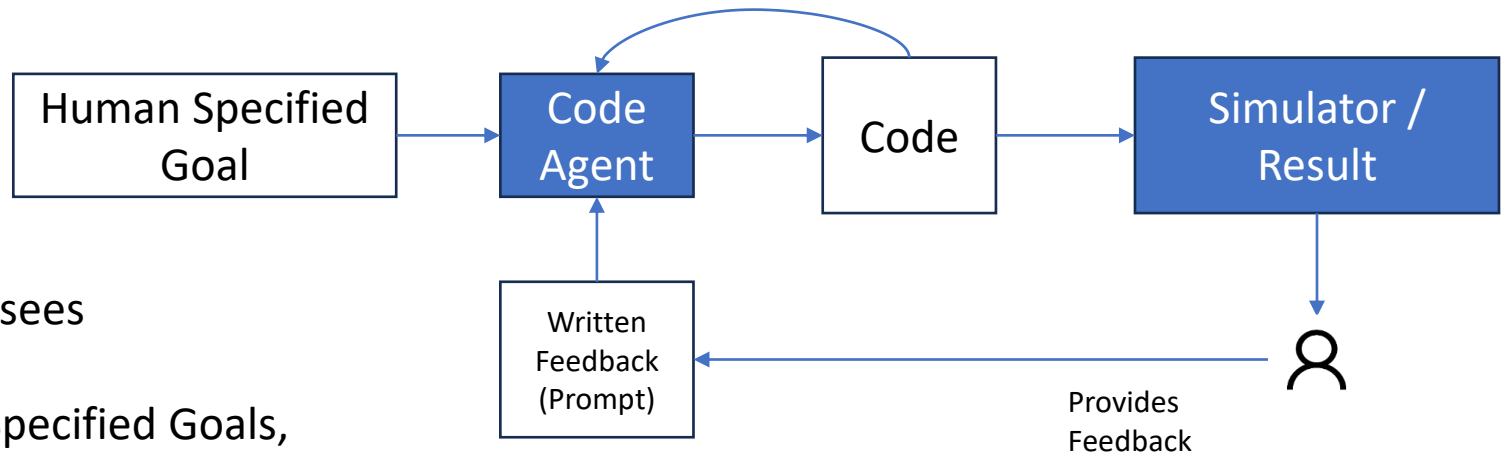
Code Agent sees

- (1) Human Specified Goals,
- (2) Execution error Trace
- (3) Current Code

Limitations of Agentic-LLM corrective feedback



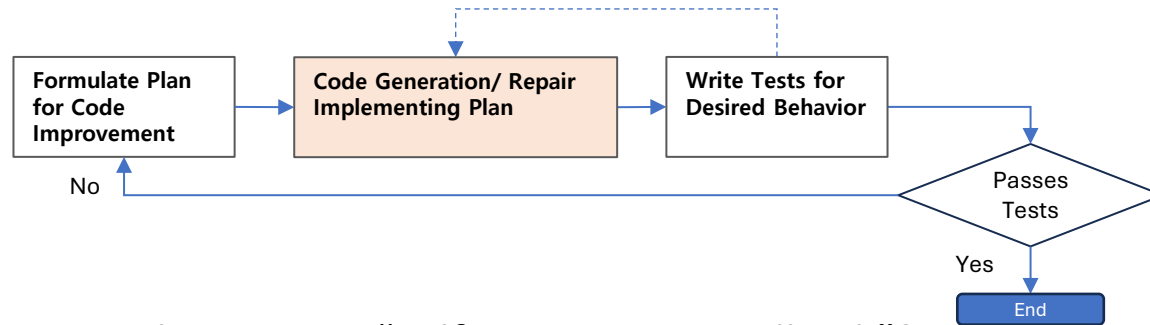
How about using “Self-Corrective Feedback”?



Code Agent sees

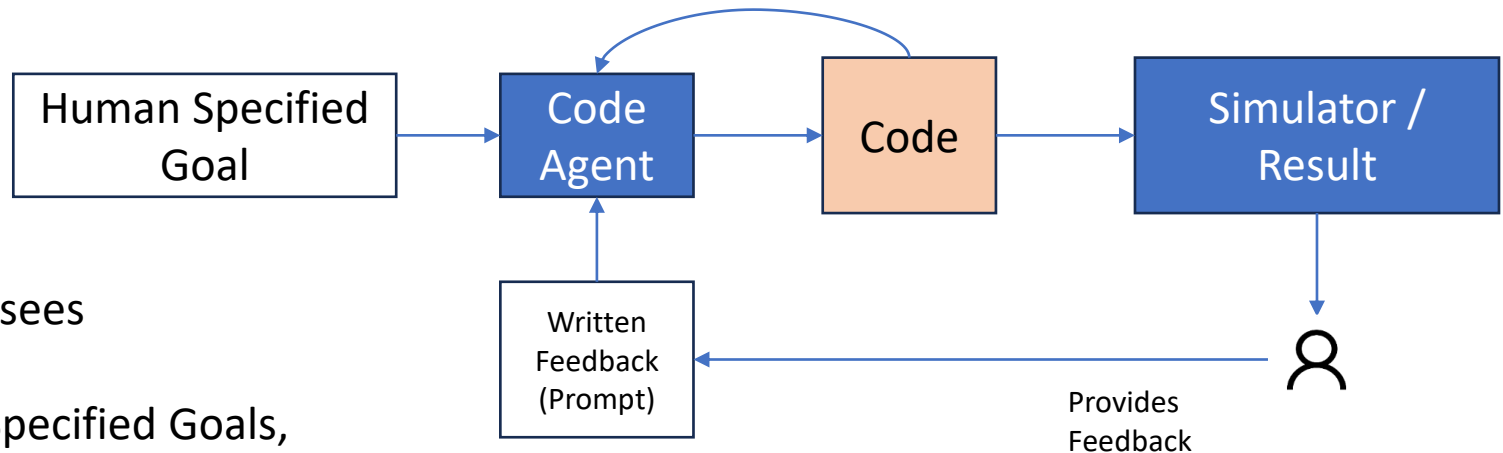
- (1) Human Specified Goals,
- (2) Current Code (no more execution Errors)
- (3) Human Feedback of the simulator results

Limitations of Agentic-LLM corrective feedback



How about using “Self-Corrective Feedback”?

(3) Hard Stuck: The code keeps getting updated, but the simulation result is the same.. **Irrecoverable Failure**



Code Agent sees

- (1) Human Specified Goals,
- (2) Current Code (no more execution Errors)
- (3) Human Feedback of the simulator results

Self-Correction can fail miserably

What the user sees:



What the User instructs:

*Rewrite the above function.
The problem with this code is
the car moves forward but
failed to make a sharp turn.*

*It should look ahead alot more,
and slow down alot more when
it detects a turn upcoming*

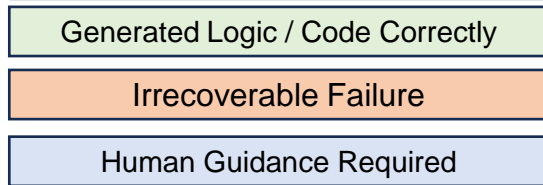
```
38     else:
39         # Slow down significantly before making a sharp turn
40         near_forward_pixel = track_mask[int(car_y + direction_vector[1] * lookahead_distance /
41         4), int(car_x + direction_vector[0] * lookahead_distance / 4)]
42         if near_forward_pixel == 0:
43             return "brake"
44         else:
45             # Check further ahead to anticipate turns earlier and stop completely if necessary
46             far_forward_pixel = track_mask[int(car_y + direction_vector[1] * lookahead_distance *
47             2), int(car_x + direction_vector[0] * lookahead_distance * 2)]
48             if far_forward_pixel == 0:
49                 return "brake"
50             else:
51                 # If a sharp turn is detected well in advance, stop completely to reassess
52                 very_far_forward_pixel = track_mask[int(car_y + direction_vector[1] *
53                 lookahead_distance * 3), int(car_x + direction_vector[0] * lookahead_distance * 3)]
54                 if very_far_forward_pixel == 0:
55                     return "brake"
56                 else:
57                     return "gas"
```

Code LLM is trying Really hard. But the same exact error keeps happening.

→ **The Key Problem:** The code generation model has no idea about what it is doing in the actual environment. The only feedback is from the human.

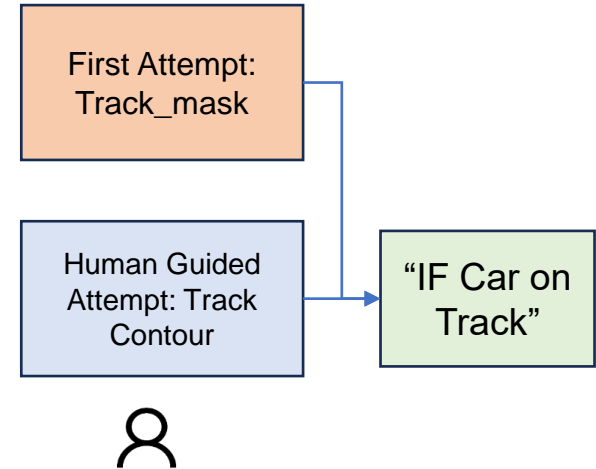
But the human has no idea what's wrong with the code. It's a **blind leading the blind situation**.

Irrecoverable Failures that cannot be handled by “think step-by-step” or Self-correction

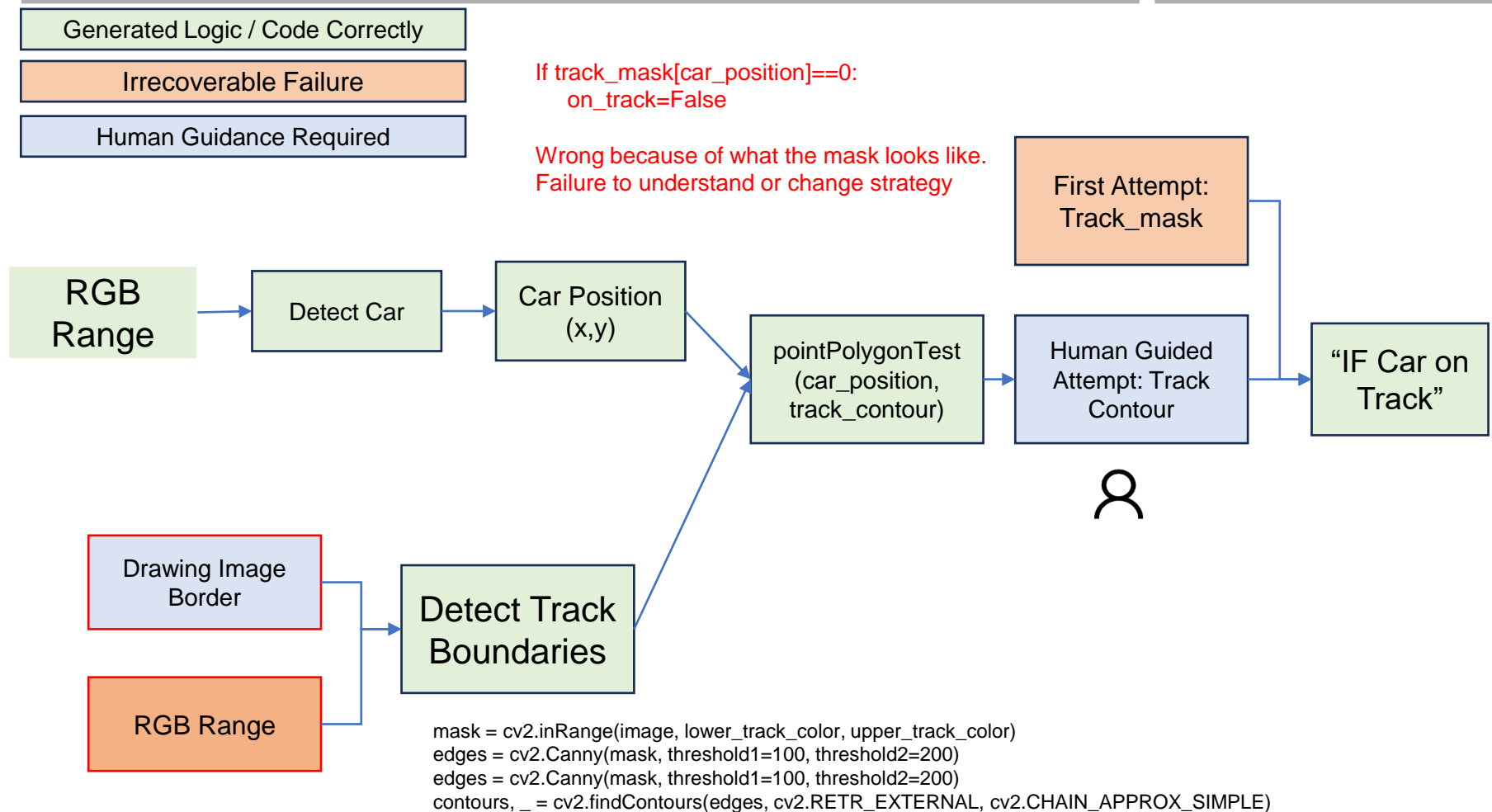


If `track_mask[car_position]==0`:
`on_track=False`

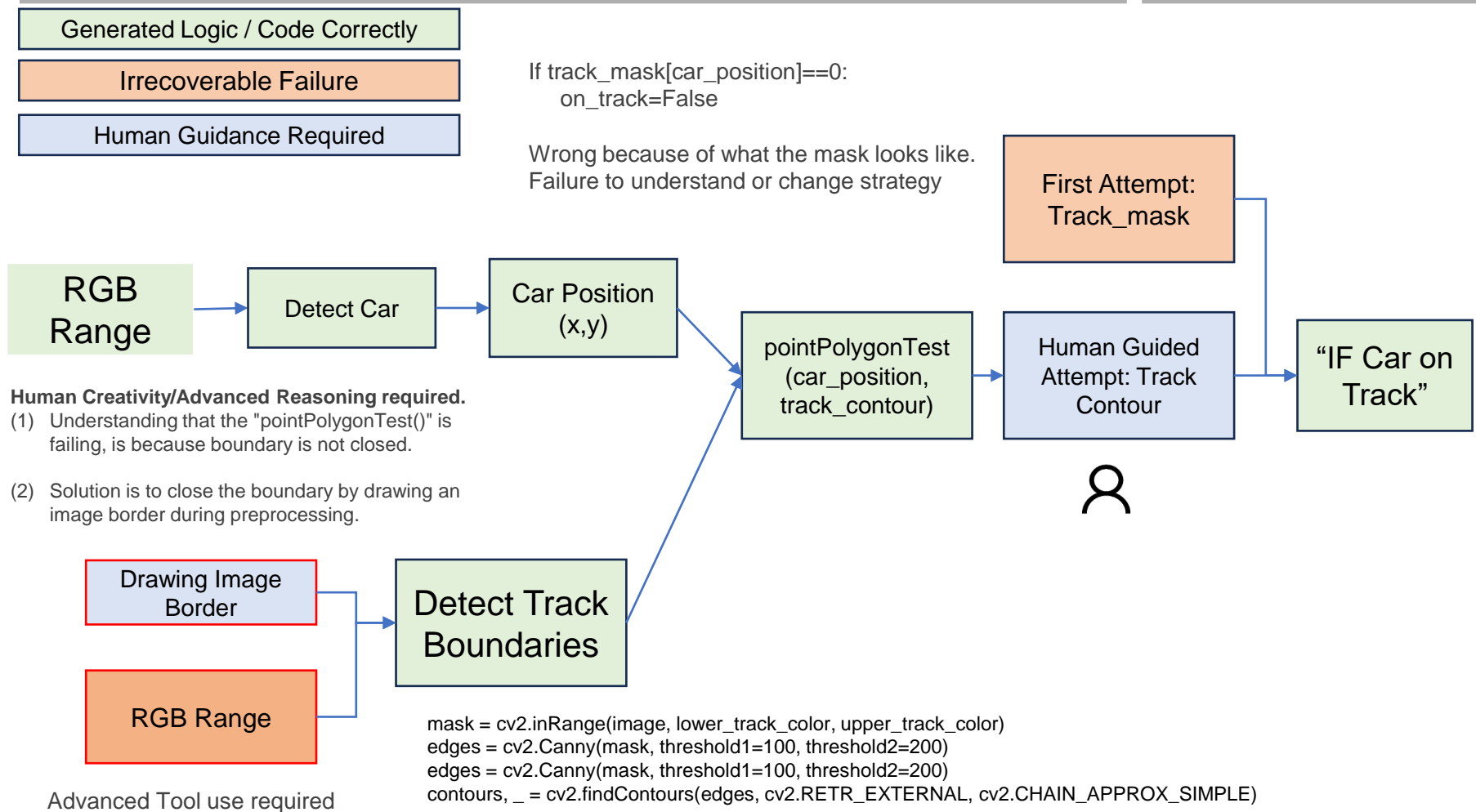
Wrong because of what the mask looks like.
Failure to understand or change strategy



Irrecoverable Failures that cannot be handled by “think step-by-step” or Self-correction



Irrecoverable Failures that cannot be handled by “think step-by-step” or Self-correction



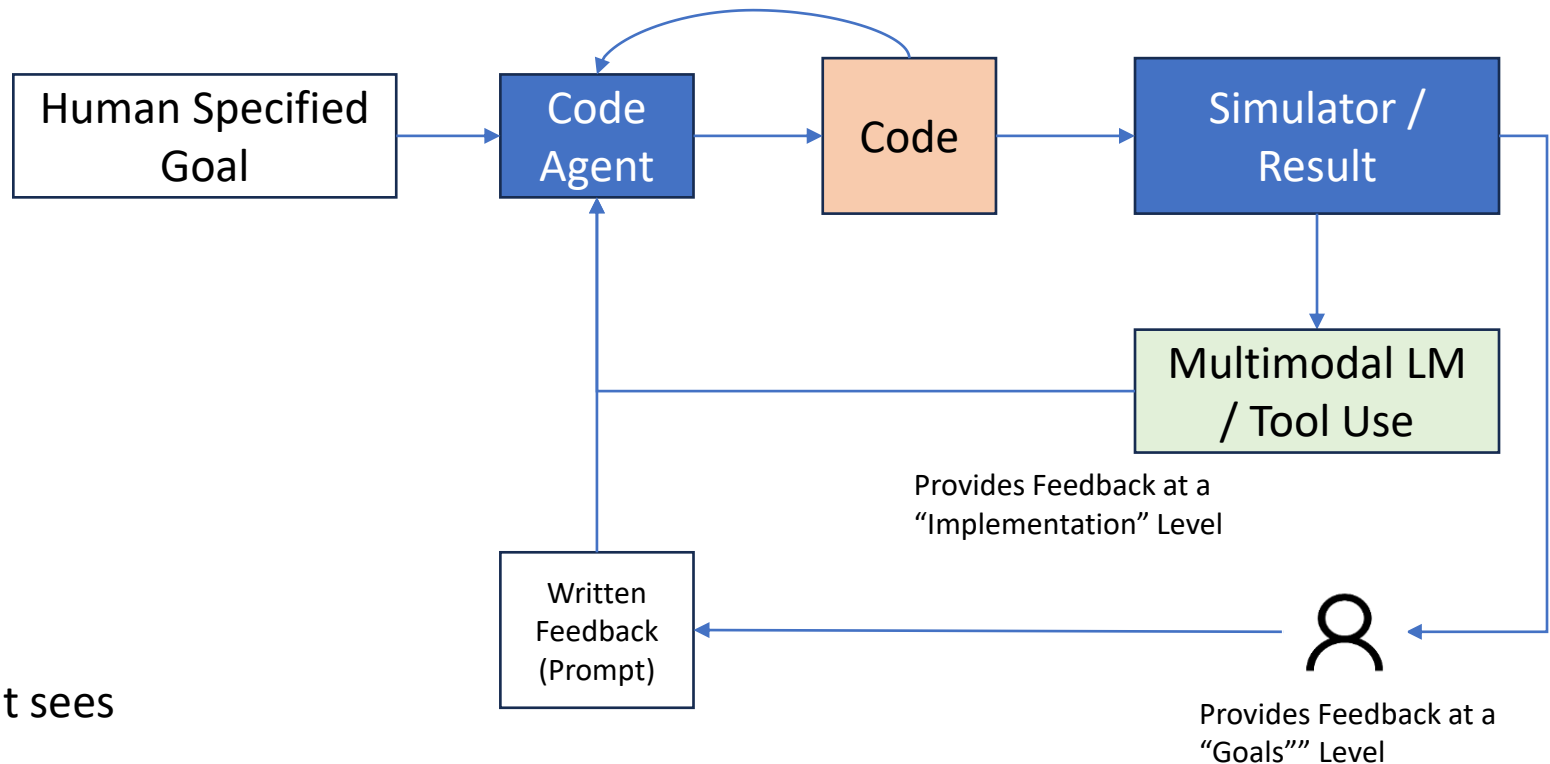
Summary

1. Steering LLMs towards Code Generation depends on the Programmer providing the relevant context to the Model, for understanding the Program's Design Principles, System Understanding and Domain Understanding.
2. Debugging and generating new code in physical control systems is extremely difficult. It's not fix patterns that you can rehash like in constructing certain types of games or UI.
3. "Formulating Plan for Code Improvement" can generally works well at a high (and not very useful level).
4. Logic errors can be due to wrong high level plan, wrong implementation logic, wrong hyperparameters, or wrong input assumptions. Neither the code agent or Human has vision over what is happening.
5. Code Generation with Agentic self-corrective Feedback, and human feedback has irrecoverable failure.

Overcoming the Limits of Agentic-LLM corrective feedback

(3) Hard Stuck: The code keeps getting updated, but the simulation result is the same.. **Irrecoverable Failure**

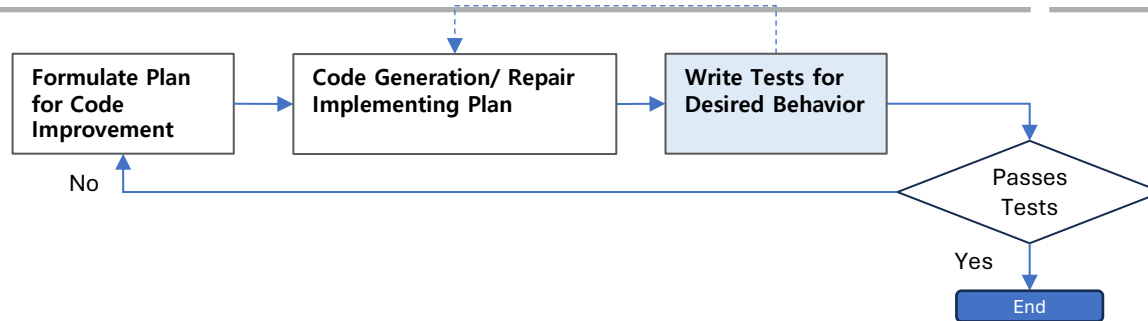
SOLUTION: Introduce Corrective Feedback with External Help



Code Agent sees

- (1) Human Specified Goals,
- (2) Current Code (no more execution Errors)
- (3) Human Feedback of the simulator results
- (4) Reconstructed "Image" of the Situation

Step 3: Writing Tests for Desired Behavior



Assumption: We can keep decomposing until task complexity is low enough, such that the Multimodal LM $p(y|x)$ can handle it; potentially with prompt examples or zero-shot



"Is the car on-track, off-track, or spiraling? Answer with only one of the three options."



Vision LM

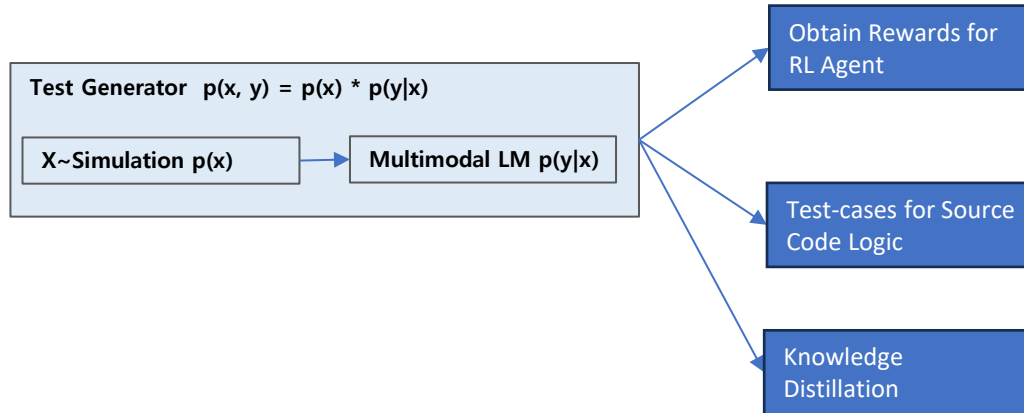
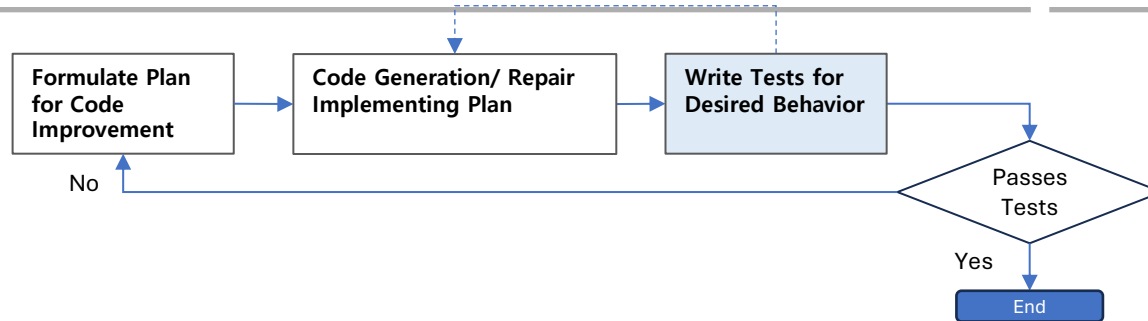


{On-track,
close-to-edge,
Off-center-left,
Off-center-right,
Off-track,
Left-turn Upcoming
Right-turn Upcoming
Spiralling,}

Why not just use a Computer Vision Model?

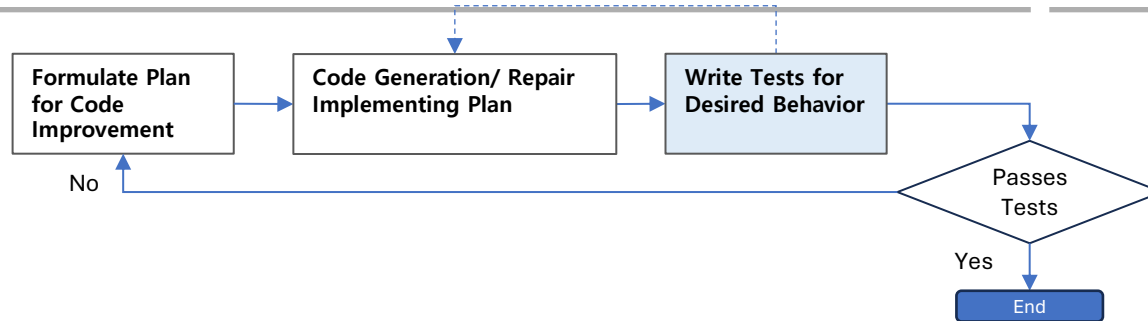
In the factory, spatio-temporal data from logs, positional coordinates, allows us to reconstruct the "image" of what happened and the sequence of events.

Step 3: Writing Tests for Desired Behavior



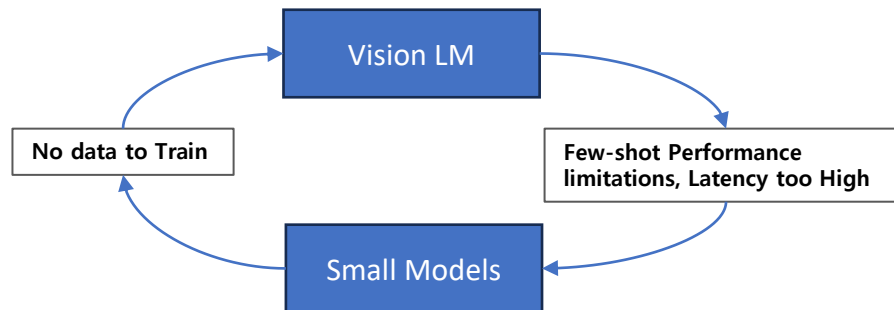
In theory, if we had a way to generate Test cases that cover the distribution of inputs and correct labels, that means we have the ability to generate synthetic data to do many things (RL rewards, training data, knowledge distillation...)..... Sounds too good to be true.

Step 3: Writing Tests for Desired Behavior



Assumption: We can keep decomposing until task complexity is low enough, such that the Multimodal LM $p(y|x)$ can handle it; potentially with prompt examples or zero-shot

Problem: We also can't keep decomposing Tasks: Inference Speed, LLM costs.



We need to address the **latency of large models in generating synthetic Tests and Labels.**

(especially if we do very granular decomposition of tasks)

Summary

1. Steering LLMs towards Code Generation depends on the Programmer providing the relevant context to the Model, for understanding the Program's Design Principles, System Understanding and Domain Understanding.
2. Debugging and generating new code in physical control systems is extremely difficult. It's not fix patterns that you can rehash like in constructing certain types of games or UI.
3. "Formulating Plan for Code Improvement" can generally works well at a high (and not very useful level).
4. Logic errors can be due to wrong high level plan, wrong implementation logic, wrong hyperparameters, or wrong input assumptions. Neither the code agent or Human has vision over what is happening.
5. Code Generation with Agentic self-corrective Feedback, and human feedback has irrecoverable failure.
6. Vision LMs can Providing Feedback (Labels) on the correctness of small functions, but we need to address the latency of Vision LMs for synthetic generation.

“Can I use LLMs for Code Generation in my Organisation”

Code Generation (actually Code Repair) is Problem-solving in Software Systems. Whether the effort is successful depends on

- Constraints that the Code needs to operate in.
- Expectations of the end-users.
- Quality of the current source code.
- Availability of well written documentation or APIs.
- Ability of Programmers themselves to steer the LLM based on strong technical knowledge.
- Mature tech organization with controls over code quality of LLM generated code to be checked in.