PROMPTING STRATEGY

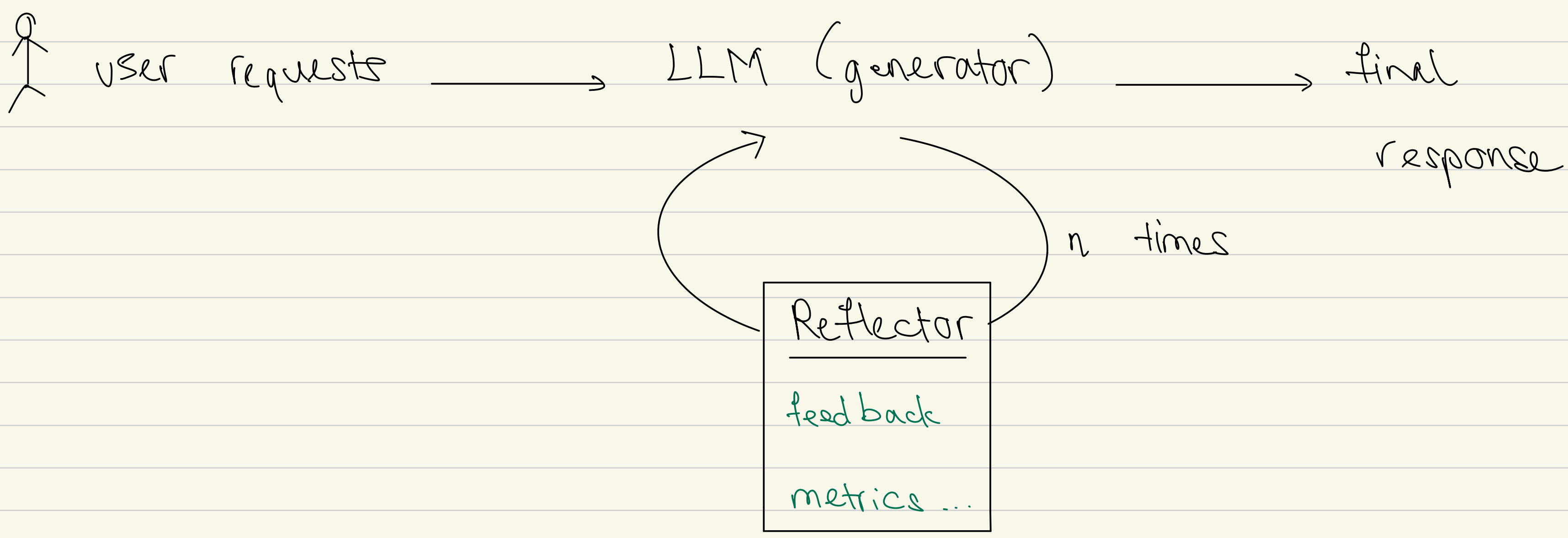
System 1 vs System 2 thinking

Very much similar concept to "Thinking: Slow and Fast" book from Malcomm Gladwell, where:

- System 1: Fast, reactive, subconscious
- System 2: Slow, reflective, conscious

Reflector

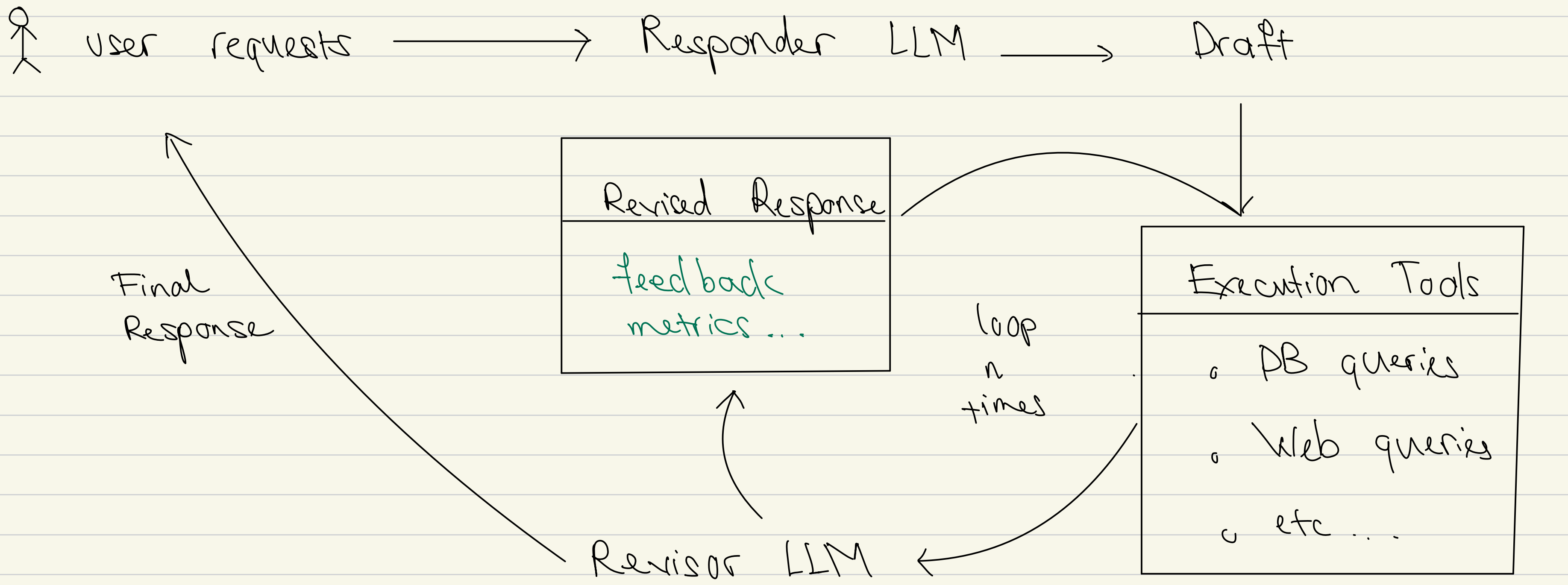
An extra step after getting response from the LLM, acts as a "critic" for the initial response. This process can happens in a loop of n times:



Cons: One major con of this simple system is the reflector does not critics based on any external data, which won't improve hallucinations much.

Reflexion:

Framework proposed by N. Shinn et al. to reinforce language agents not by updating weights, but instead through linguistic feedback.



Language Agent Tree Search

Mote Carlo Tree Search (MCTS)

When to use MCTS ?

When a decision tree of a problem has a large number of branches that makes the brute force solution (explore all the possible branches) too costly. The time complexity of such solution is usually $O(h^w)$, where h is the decision tree's height and w is the decision tree's width (branches) at each node.

→ MCTS offers a balanced approach between Exploration and Exploitation

How MCTS works ?

MCTS

```
root_node
For n of simulations:
    node = selection(root_node)
    score = playout(node)
    backpropagate(node, score)
return best_child(root_node)
```

Purpose: Pick the best action from root

Complexity: $O(\text{simulations} * (\text{selection} + \text{playout} + \text{backprop}))$

Selection (node)

```
while node is not terminal:
    if node can expand:
        return expand(node)
    else:
        node = best_child(node)
return node
```

Purpose: Select a node to playout from. It's either:

- Terminal node
- Newly expanded node

Expand(node)

```
action = pick_untried_actions()
next_state = Game.perform(action)
node.children.append(new Node(next_state))
```

Purpose: Explore more outcomes

best_child (node)

Pick the child with highest score using this formula:

$$\frac{child.score}{child.visits} + C\sqrt{\frac{2\ln(parent.visits)}{child.visits}}$$

playout (node)

```
curr_state = node.state
while curr_state is not Game.is_over():
    next_action = pick_action_from_state(curr_state)
    curr_state = Game.perform(next_action)
return Game.score()
```

Purpose: Play out a game until the end, return the score

backpropagate (node, score)

```
while node is not None:
    node.score += score
    node.visits += 1
    node = node.parent
```

Purpose: Update the score of all nodes on the path lead to this outcome

Language Agent Tree Search (LATS) (Zhou et al)

The paper uses MCTS to reduce LLM's hallucination. The overall structure of LATS is the same as MCTS, with the difference in "playout" step.

LATS's playout step

```
actions_sequence = generate_actions_sequence_LLM (node.state)
final_state = simulate_sequence (node.state, action_sequence)
return evaluate_outcome (final_state)
```