```
# AI Decision-Making System

import json
import time

MEMORY_FILE = "ai_memory.json"

class AIDecisionMaker:
    def __init__(self):
        self.memory = self.load_memory()

    def load_memory(self):
        try:
            with open(MEMORY_FILE, "r") as f:
                return json.load(f)
        except FileNotFoundError:
            return {"modifications": [], "decisions": []}

    def save_memory(self):
        with open(MEMORY_FILE, "w") as f:
            json.dump(self.memory, f, indent=4)

    def evaluate_past_optimizations(self):
        print("\nEvaluating Past Optimizations...")
        if not self.memory["modifications"]:
            print("No past modifications found.")
            return

        scores = {}
        for mod in self.memory["modifications"]:
            function = mod["function_name"]
            performance_gain = mod.get("performance_gain", 1)
            scores[function] = scores.get(function, 1) * performance_gain

        self.memory["scores"] = scores
        self.save_memory()
        print("Optimization Evaluation Complete.")

    def select_next_optimization(self):
        if not self.memory["scores"]:
            print("No performance scores found. Running default strategy.")
            return "default_function"

        best_function = max(self.memory["scores"], key=self.memory["scores"].get)
        print(f"AI Selected: {best_function} for next optimization.")
        return best_function

    def run(self):
        self.evaluate_past_optimizations()
        next_function = self.select_next_optimization()

        decision_entry = {
            "timestamp": time.strftime("%Y-%m-%d %H:%M:%S"),
            "selected_function": next_function
```

```
        }

        self.memory["decisions"].append(decision_entry)
        self.save_memory()

           print("\nAI Decision-Making Complete! AI has selected the next function for
improvement.")
        return next_function

if __name__ == "__main__":
    ai_decider = AIDecisionMaker()
    ai_decider.run()
```