# Weather and Solar Forecast AI - Project Report

## 1. GPT-4 Integration and Function Calling

### A. Detailed Function Definitions (main.py)

* analyze\_intent\_from\_user\_input(user\_input: str) -> str  
  Classifies user query as weather, marine, air\_quality, or general.
* select\_function\_based\_on\_intent(intent: str) -> Callable  
  Matches intent to appropriate handler function (e.g., fetch\_weather\_data).
* invoke\_function(func: Callable, params: dict) -> dict  
  Invokes function with parameters extracted from GPT or default values.

### B. GPT Call Example (With Function Calling Support)

response = await client.chat.completions.create(  
 model="gpt-4",  
 messages=messages,  
 tools=[  
 {  
 "type": "function",  
 "function": {  
 "name": "fetch\_weather\_data",  
 "description": "Gets solar and weather forecast data for a location",  
 "parameters": {  
 "type": "object",  
 "properties": {  
 "location": {"type": "string"},  
 "days": {"type": "integer"}  
 },  
 "required": ["location"]  
 }  
 }  
 }  
 ],  
 tool\_choice="auto"  
)

## 2. Enhanced Moderation System

### A. Moderation Integration (main.py)

* Uses openai.Moderation.create(input=user\_input)
* Checks flags: hate, violence, self-harm, sexual, harassment
* Logs flagged cases with datetime, user, input, flagged\_category

### B. Flow

[User Input]  
 ↓  
[Preprocessing (strip, clean, profanity filter)]  
 ↓  
[OpenAI Moderation API]  
 ↓ ↘  
 Safe Flagged  
 ↓ ↓  
 Proceed Show warning

## 3. Advanced Weather Data Processing

### A. Multi-API Support (main.py)

* Open-Meteo: Primary source for hourly/daily irradiance
* Solcast (optional): Supports global tilted irradiance, cloud opacity
* API selection logic: Falls back to alternate providers on failure or low confidence

### B. Example

if provider == "open-meteo":  
 return fetch\_from\_open\_meteo(lat, lon)  
elif provider == "solcast":  
 return fetch\_from\_solcast(lat, lon)

## 4. Enhanced Error Handling System

### A. Types of Errors Handled

* API connection errors (httpx.ConnectTimeout)
* JSON decode issues
* GPT response parsing errors
* Parameter extraction failures

### B. Error Flow

try:  
 result = fetch\_weather\_data()  
except httpx.TimeoutException:  
 return {"error": "API timeout. Please try again later."}

## 5. Advanced Caching System

### A. Purpose

* Avoid redundant API calls
* Improve GPT response time for repeated queries

### B. Implementation

* In-memory (e.g., functools.lru\_cache)
* Optional: Redis for persistent cache

@lru\_cache(maxsize=128)  
def get\_cached\_weather(location: str):  
 return fetch\_weather\_data(location)

## 6. Performance Monitoring System

### A. Metrics Logged

* GPT response time
* Weather API latency
* Moderation hit rate
* Function call success/failure

### B. Logging Example

logger.info(f"Function {func.\_\_name\_\_} took {elapsed:.2f} seconds")

## ✅ Summary

This extended technical section:

* Makes your project future-proof with maintainable structure
* Demonstrates thoughtful architecture with safety, modularity, and performance in mind
* Aligns with OpenAI best practices in moderation and function calling

## 📌 Continuous Forecasting Capability in Applications

### A. Overview

The Solar Forecast AI system supports **continuous solar forecast extraction** via:

* Scheduled querying (e.g., every 1–6 hours)
* UI-based refresh on demand
* External automation tools (e.g., CRON, background workers)

### B. How LLM Assists

* **Parameter awareness**: GPT understands location, duration, intent
* **Function loop**: Forecast function can be called programmatically in intervals
* **Stateful context (optional)**: Retains user preferences during session

### C. Implementation Example

import asyncio  
  
async def continuous\_forecast(location: str, interval\_minutes: int):  
 while True:  
 data = fetch\_weather\_data(location)  
 print(data)  
 await asyncio.sleep(interval\_minutes \* 60)  
  
# Run every 3 hours  
asyncio.run(continuous\_forecast("Bengaluru", 180))

### D. Integration Points

* UI polling with React timers or WebSockets
* Backend CRON jobs
* Azure/AWS scheduled functions

### E. Use Cases

* Solar energy dashboards
* Agricultural planning
* IoT-based smart grid adjustments

## 🗂 System Architecture Overview

graph TB  
 subgraph "Frontend"  
 UI["React UI"]  
 Redux["Redux State Management"]  
 UI --> Redux  
 end  
   
 subgraph "Backend"  
 FastAPI["FastAPI Server"]  
 ML["ML Model Service"]  
 DB[(Database)]  
 FastAPI --> ML  
 FastAPI --> DB  
 end  
   
 subgraph "Docker Infrastructure"  
 Docker["Docker Compose"]  
 Docker --> |Orchestrates| Frontend  
 Docker --> |Orchestrates| Backend  
 end  
   
 UI --> |API Requests| FastAPI  
   
This visualizes how your **React frontend**, **FastAPI backend**, **ML services**, and **infrastructure orchestration with Docker** interact.