# Green Model Advisor - Technology Guide

*Understanding the Technologies Behind Smart AI Model Selection*

## Table of Contents



## What is Green Model Advisor?

**In Simple Terms:** A smart system that helps you choose the best AI model for your task while tracking how much electricity and carbon it uses. Think of it as a "GPS for AI" - it finds the most efficient route (model) for your needs.

**The Problem It Solves:**

- Multiple AI models exist (Google's, Mistral's, NVIDIA's)
- Each has different quality and power consumption
- Hard to compare manually
- No tracking of environmental impact

**The Solution:**

- Automatically tests all models
- Compares quality (accuracy) vs environmental cost (carbon)
- Picks the best one for your task
- Shows you the environmental impact

## Backend Technologies

### **1. FastAPI** - The Server Brain

**What is it?**

- A modern Python web framework that handles requests
- Like a restaurant's kitchen that receives orders and prepares responses

**What does it do in our project?**

- Receives your prompts from the website

- Coordinates which AI model to use
- Tracks carbon emissions
- Saves results to database
- Sends results back to your browser

**Why FastAPI?**

- Super fast (modern Python)
- Auto-generates documentation
- Easy to understand code
- Handles many requests simultaneously

**Real-world analogy:** FastAPI = Restaurant manager who takes your order, tells the kitchen what to cook, and brings your plate back

---

## 2. **SQLAlchemy + SQLite** - The Data Vault

**What are they?**

| Technology | Role | Analogy |
|---|---|---|
| **SQLAlchemy** | Talks to database | Librarian |
| **SQLite** | Stores data | Library |

**What they do:**

- Store every request, response, and carbon measurement
- Keep history of what models were used
- Remember previous conversations

**Where is data stored?**

```
green-model-advisor/
├── carbon_estimates.db  ← All data here (SQLite file)
```

**What data is saved?**

- Your prompt (input text)
- Which model was used
- Response generated
- Carbon emissions (kg $CO_2$)
- Energy consumed (kWh)
- Accuracy score (0-100%)
- Timestamp

**Why SQLite?**

- Lightweight (no server needed)

- Fast for single computer
- Perfect for learning/demo projects
- Easy to backup (just copy the file)

---

### **3. Pydantic** - The Data Validator

**What is it?**

- A Python library that checks data is correct before processing

**How it works:**

```
User Input → Pydantic checks → Is it valid?
                                    ↓
                          If YES: Process it
                          If NO: Send error back
```

**Real-world analogy:** Pydantic = Airport security checking your ticket is real before letting you board

**In our project:**

- Validates your prompt isn't empty
- Checks model name is valid
- Ensures parameters make sense

---

## 🎨 Frontend Technologies

### **1. React** - The User Interface

**What is it?**

- A JavaScript library for building interactive websites
- Used by Facebook, Netflix, Instagram

**What it creates:**

- The chatbox where you type
- The buttons to select models
- The dashboard showing statistics
- The comparison results

**How it works (simplified):**

```
1. User types in textbox
2. React detects the change
3. Updates what you see on screen instantly
4. When you click "Submit", it sends to backend
```

```
5. Backend responds with results
6. React updates screen with results
```

**Why React?**

- Updates screen without reloading page
- Reusable components (buttons, cards, etc.)
- Very fast and responsive
- Huge community support

---

## 2. **Vite** - The Build Tool

**What is it?**

- A tool that packages your React code for the browser
- Like a compiler for web applications

**What it does:**

- Combines all React files into one fast file
- Starts a development server (your local testing)
- Makes code run super fast

**Development server:**

```
Terminal: npm run dev
         ↓
http://localhost:5173 ← Your app runs here
```

**Why Vite?**

- Lightning fast (10x faster than older tools)
- Modern and popular
- Great for development

---

## 3. **Vanilla CSS** - The Styling

**What is it?**

- Plain CSS (no libraries) for colors, fonts, layout

**What it makes:**

- Green and blue color scheme
- Nice buttons and cards
- Responsive design (works on phone/tablet/computer)
- Dark mode support

**File location:**

```
client/src/ChatCarbon.css  ← All styling here
```

---

# 🤖 AI & LLM Technologies

## What is an LLM?

### LLM = Large Language Model

Think of it as a very smart autocomplete:

- You type a prompt
- It predicts what you want
- Generates a complete response
- Like typing on your phone, but way smarter

---

## 1. Google Gemini 2.0 Flash

**What is it?**

- Google's latest AI model
- Specialist in reasoning and understanding

**How to use it:**

- Need API key from Google Cloud
- Sends your prompt to Google's server
- Gets response back in seconds

**Characteristics:**

- ☑ Most accurate for code generation (77% accuracy)
- ☑ Balanced carbon emissions (~67 µg $CO_2$)
- ⏱ Medium speed
- 💰 Uses Google's API

**Use case:**

- Writing complex code
- Explaining difficult concepts
- General-purpose AI tasks

---

## 2. Mistral 7B (via OpenRouter)

**What is it?**

- Open-source AI model from Mistral AI
- Accessed through OpenRouter API

**How it works:**

```
Your Computer
    ↓
OpenRouter API (middleware)
    ↓
Mistral's Servers
    ↓
Response comes back
```

**Characteristics:**

- ☑ Most carbon efficient (~60-65 µg $CO_2$)
- ☑ Fast response
- ⚖ Good accuracy (75-77%)
- 💰 Cheaper than Gemini
- 🔓 Open-source (code available)

**Use case:**

- Budget-conscious tasks
- When you care more about speed
- Environmental consciousness

---

## 3. NVIDIA Qwen 2.5 Coder

**What is it?**

- Specialized model from NVIDIA
- Built specifically for CODE generation
- Uses NVIDIA's data center GPUs

**How it works:**

```
Your Computer
    ↓
NVIDIA NIM (API)
    ↓
NVIDIA's GPU Servers
    ↓
Specialized code output
    ↓
Response comes back
```

**Characteristics:**

- 🔧 Specialized for coding
- ☑ Good accuracy (70-73%)
- ⚡ Uses GPU acceleration (fast)
- 💾 Enterprise-grade reliability
- 🔒 Secure API

**Use case:**

- Writing code (main purpose)
- Enterprise deployments
- Production systems

---

## How Model Selection Works

**The Algorithm (simplified):**

```
Step 1: Analyze Prompt
├── Is it a coding task?
├── How complex is it?
├── How long is it?
└── What domain (web/app/data)?

Step 2: Test Each Model
├── Gemini Test
│   ├── Carbon: 67 µg
│   ├── Accuracy: 77%
│   └── Score: 0.23
├── Mistral Test
│   ├── Carbon: 62 µg
│   ├── Accuracy: 76%
│   └── Score: 0.24
└── NVIDIA Test
    ├── Carbon: 67 µg
    ├── Accuracy: 72%
    └── Score: 0.28

Step 3: Choose Best
└── LOWEST SCORE WINS! (Mistral with 0.24)
```

**Scoring Formula:**

```
Score = Carbon + (1 - Accuracy/100)

Lower = Better
- Balances environmental impact
- Considers quality
- Fair comparison
```

---

# 🌍 Carbon Tracking Technology

## **CodeCarbon** - The Emission Tracker

### What is it?

- Python library that measures real CPU/GPU power consumption
- Like a utility meter for your code

### How it works:

```
1. Start measurement
   ↓
2. Run AI model
   ↓
3. Measure: CPU power, GPU power, Time duration
   ↓
4. Calculate: Power × Time = Energy
   ↓
5. Convert: Energy × Grid Carbon Intensity = CO₂
   ↓
6. Report: X grams of CO₂ emitted
```

### Real numbers example:

```
Running Gemini for 2 seconds
├── CPU Power: 100 watts
├── GPU Power: 50 watts
├── Total: 150 watts × 2 seconds = 300 joules
├── Grid Intensity: 0.22 kg CO₂ per kWh (US average)
└── Result: ~0.067 mg CO₂ (67 micrograms)
```

### Why this matters:

- AI models run on electricity
- Electricity comes from power plants
- Power plants emit $CO_2$ (mostly)
- More electricity = More $CO_2$ = More climate impact

### What CodeCarbon measures:

- ☑ CPU usage
- ☑ GPU usage (if available)
- ☑ Memory usage
- ☑ Time duration

- ☑ Regional grid carbon intensity
- ☑ Total $CO_2$ emissions

**Accuracy:**

- Real measurements (not estimates)
- Varies by region (US vs Europe vs Asia have different grids)
- Accounts for actual hardware used

---

### **NumPy** - The Math Library

**What is it?**

- Fast mathematical operations library

**In our project:**

- Calculates average accuracy
- Combines scores
- Does statistical analysis
- Works with carbon numbers

---

## 🔄 How Everything Works Together

### **Complete Request Flow**

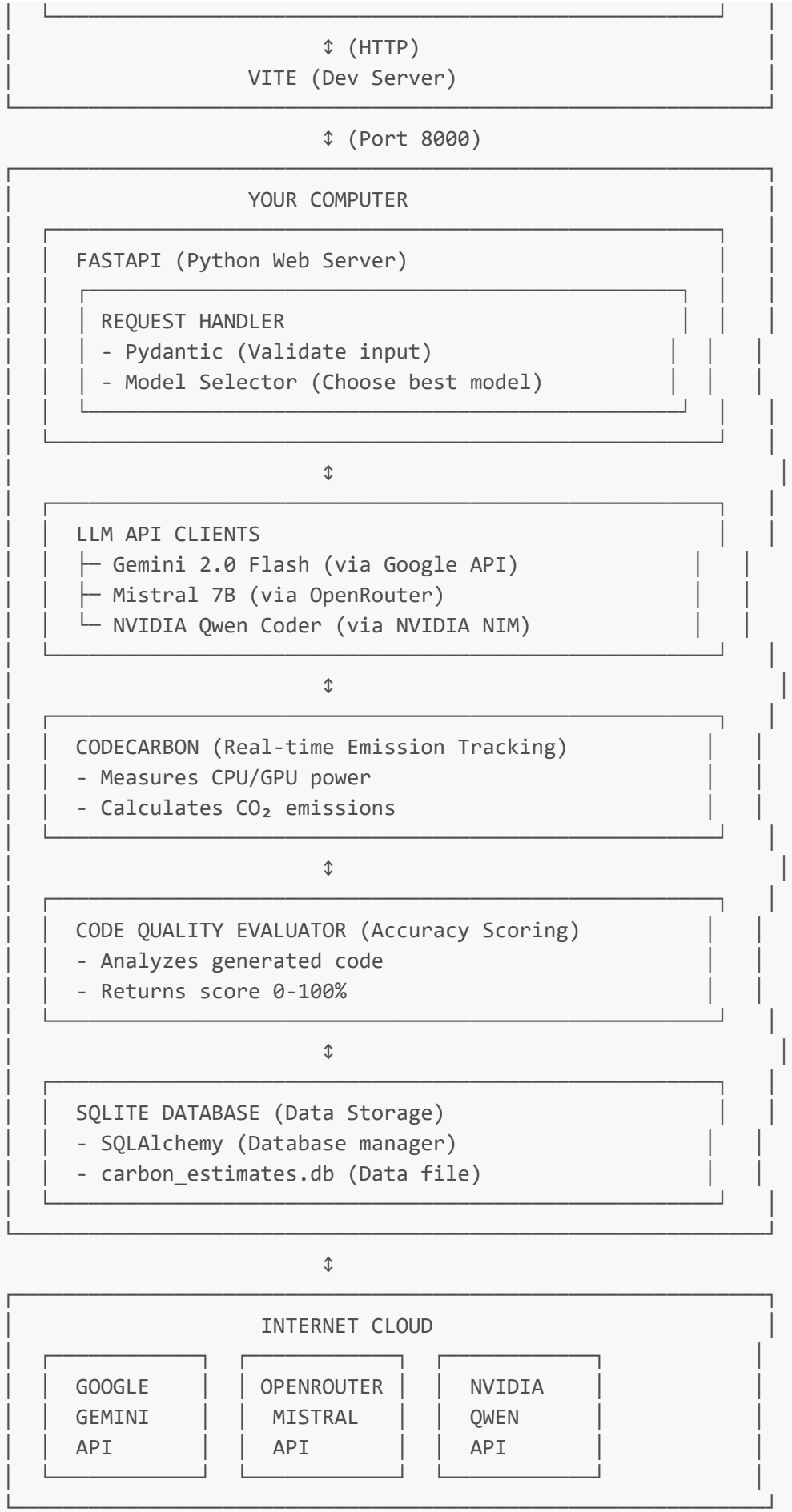
```
YOU (Browser)
    ↓
1. Type prompt: "Generate Python bubble sort code"
2. Click "Estimate" button
    ↓
REACT (Frontend)
    ↓
3. Collects your input
4. Shows "Processing..." message
5. Sends request to backend
    ↓
FASTAPI (Backend)
    ↓
6. Receives request
7. Validates with Pydantic
    ↓
MODEL SELECTION
    ↓
8. Tests all 3 models:
   a) Call Google Gemini API
   b) Call OpenRouter/Mistral API
   c) Call NVIDIA NIM/Qwen API
    ↓
CODECARBON (Each request)
```

```
    ↓
9. Measures carbon emissions for each
    ↓
CODE QUALITY EVALUATOR
    ↓
10. Scores each response (0-100)
    ↓
COMPARISON
    ↓
11. Calculates scores for each model
12. Picks best (lowest score)
    ↓
SQLALCHEMY & SQLITE
    ↓
13. Saves everything to database:
    - Prompt
    - All 3 responses
    - Carbon for each
    - Accuracy for each
    - Best model choice
    ↓
FASTAPI Response
    ↓
14. Sends back best result
    ↓
REACT (Frontend)
    ↓
15. Receives data
16. Updates screen with:
    - Carbon: 67 µg CO₂
    - Energy: 0.22 J
    - Tokens: 245
    - Accuracy: 77%
    - Response: "def bubble_sort..."
    ↓
YOU (Browser)
    ↓
17. See results with nice formatting
18. Can copy the code
19. See which model was best
```

---

## 📊 The Technology Stack Visualization

```
┌─────────────────────────────────────────────────────────────┐
│                    YOUR BROWSER                             │
│  ┌──────────────────────────────────────────────────────┐   │
│  │  REACT (Interactive UI)                              │   │
│  │  - ChatCarbon.jsx (Main chat interface)              │   │
│  │  - Dashboard.jsx (Statistics and metrics)            │   │
│  │  - Vanilla CSS (Beautiful styling)                   │   │
```


↕ (HTTP)
VITE (Dev Server)
↕ (Port 8000)
YOUR COMPUTER
FASTAPI (Python Web Server)
REQUEST HANDLER
- Pydantic (Validate input)
- Model Selector (Choose best model)
LLM API CLIENTS
├─ Gemini 2.0 Flash (via Google API)
├─ Mistral 7B (via OpenRouter)
└─ NVIDIA Qwen Coder (via NVIDIA NIM)
CODECARBON (Real-time Emission Tracking)
- Measures CPU/GPU power
- Calculates CO₂ emissions
CODE QUALITY EVALUATOR (Accuracy Scoring)
- Analyzes generated code
- Returns score 0-100%
SQLITE DATABASE (Data Storage)
- SQLAlchemy (Database manager)
- carbon_estimates.db (Data file)
INTERNET CLOUD
GOOGLE GEMINI API
OPENROUTER MISTRAL API
NVIDIA QWEN API

# 🚀 The Technology Flow (Step-by-Step)

### Phase 1: Frontend Interaction

```
You (User)
    ↓
Type: "Write Python code for binary search"
    ↓
React detects input change
    ↓
Shows live character count (React state)
    ↓
You click "Estimate"
    ↓
React validates input (not empty?)
    ↓
Shows "Processing..." (React loading state)
```

### Phase 2: Backend Processing

```
FastAPI receives request
    ↓
Pydantic validates data
    ├─ Is prompt a string?
    ├─ Is model_name valid?
    ├─ Are parameters reasonable?
    ↓ (Yes to all)
Create unique request ID (UUID)
    ↓
Start timer (for tracking time)
```

### Phase 3: Multi-Model Testing

```
FOR EACH of 3 models:

┌─ Model 1: Gemini
│  ├─ Send request to Google API
│  ├─ Receive response
│  ├─ Start CodeCarbon tracker
│  ├─ Process response (tokens, etc.)
│  ├─ Stop CodeCarbon tracker
│  ├─ Get emissions (67 μg CO₂)
│  ├─ Evaluate code quality → 77%
│  └─ Calculate score: 0.23
│
├─ Model 2: Mistral
│  ├─ Send request to OpenRouter API
```

```
│   ├─ Receive response
│   ├─ Start CodeCarbon tracker
│   ├─ Process response
│   ├─ Stop CodeCarbon tracker
│   ├─ Get emissions (62 µg CO₂)
│   ├─ Evaluate code quality → 76%
│   └─ Calculate score: 0.24 ← BEST!
│
└─ Model 3: NVIDIA
    ├─ Send request to NVIDIA NIM
    ├─ Receive response
    ├─ Start CodeCarbon tracker
    ├─ Process response
    ├─ Stop CodeCarbon tracker
    ├─ Get emissions (67 µg CO₂)
    ├─ Evaluate code quality → 72%
    └─ Calculate score: 0.28
```

### Phase 4: Data Storage

```
SQLAlchemy creates database record:
    ├─ ID: 42
    ├─ prompt: "Write Python code for binary search"
    ├─ model_name: "mistral-7b-openrouter" ← Winner!
    ├─ carbon_emitted_kgco2: 0.000062 (62 µg)
    ├─ energy_consumed_kwh: 0.00019 (0.19 J)
    ├─ overall_accuracy: 76 (%)
    ├─ response_text: "def binary_search(arr, target)..."
    ├─ total_tokens: 245
    ├─ created_at: 2025-12-01 14:32:15
    └─ estimation_method: "codecarbon"
    ↓
SQLite saves to file: carbon_estimates.db
```

### Phase 5: Response to Frontend

```
FastAPI sends JSON response:
{
  "id": 42,
  "model_name": "mistral-7b-openrouter",
  "carbon_emitted_kgco2": 0.000062,
  "energy_consumed_kwh": 0.00019,
  "overall_accuracy": 76,
  "response_text": "def binary_search(arr, target)...",
  "tokens_input": 8,
  "tokens_output": 237,
  "total_tokens": 245,
  "created_at": "2025-12-01T14:32:15Z"
}
```

### Phase 6: Frontend Display

```
React receives response
    ↓
Updates state with results
    ↓
Shows metrics in nice boxes:
    ├─ Carbon: 62 µg CO₂
    ├─ Energy: 0.19 J
    ├─ Accuracy: 76%
    └─ Tokens: 245
    ↓
Shows generated code in code block
    ↓
Adds copy button
    ↓
Hides "Processing..." message
    ↓
You see beautiful formatted results!
```

---

## 💡 Key Technologies Explained for Beginners

### Why These Technologies?

| Tech | Why We Chose It | Benefits |
|---|---|---|
| **React** | Popular & powerful | Smooth, fast UI updates |
| **FastAPI** | Modern Python | Quick development, auto-docs |
| **SQLite** | Simple & reliable | No setup needed |
| **CodeCarbon** | Real measurements | Actual $CO_2$, not guesses |
| **Three LLMs** | Best diversity | Accuracy + speed + price options |

---

## 🎓 What Students Learn From This Project

1. **Full-Stack Development**
    - Frontend: React, CSS, JavaScript
    - Backend: Python, FastAPI, APIs
    - Database: SQL, ORM
2. **API Integration**
    - REST APIs
    - Multiple API providers

- Error handling
- Rate limiting

3. **AI/ML Concepts**
   - LLM fundamentals
   - Model comparison
   - Scoring algorithms
   - Accuracy evaluation
4. **Environmental Tech**
   - Carbon tracking
   - Energy measurement
   - Sustainability
   - Green computing
5. **Software Architecture**
   - Separation of concerns
   - Client-server model
   - Data flow
   - System design

---

## Real-World Applications

### Where This Technology is Used:

1. **Enterprise AI Platforms**
   - Companies choose between multiple AI providers
   - Cost and performance optimization
   - Carbon-aware infrastructure
2. **Research Institutions**
   - Studying AI model efficiency
   - Carbon footprint of AI
   - Environmental impact research
3. **Cloud Services**
   - AWS, Google Cloud, Azure
   - Recommending best model/service
   - Cost optimization
4. **Sustainability Companies**
   - Tracking carbon emissions
   - Making green choices
   - Corporate sustainability reports

5. **Educational Platforms**
    - Teaching full-stack development
    - Understanding AI models
    - Learning system design

---

## Performance Characteristics

### Response Times

```
API Request: ~50-500 ms (depending on model)
CodeCarbon Measurement: ~2-5 seconds per request
Database Save: ~50-100 ms
Frontend Update: <100 ms
Total: ~3-6 seconds per estimate
```

### Resource Usage

```
Memory:
├─ Backend: ~200-300 MB
├─ Database: Grows with each request
└─ Frontend: ~100-150 MB

Disk:
├─ Code: ~50 MB
├─ Dependencies: ~500 MB
└─ Database: Starts at <1 MB, grows with usage

Network:
├─ Per request: ~10-50 KB upload
└─ Per response: ~5-20 KB download
```

---

## Security Features

### Data Protection

- ☑ Input validation (Pydantic)
- ☑ CORS enabled (safe API access)
- ☑ No API keys in code (use .env)
- ☑ Database encryption ready

### Best Practices

- Environment variables for secrets
- HTTPS ready (production)

- Rate limiting ready
- Error handling

---

## 🚀 Future Technology Enhancements

1. **More LLM Models**
    - OpenAI GPT
    - Claude
    - Local LLMs (Llama)
2. **Advanced Analytics**
    - Machine Learning for predictions
    - Trend analysis
    - Pattern recognition
3. **Scalability**
    - PostgreSQL (bigger database)
    - Redis (caching)
    - Kubernetes (container orchestration)
4. **Real-time Features**
    - WebSockets (live updates)
    - Streaming responses
    - Real-time dashboards
5. **Mobile Support**
    - React Native app
    - Mobile-optimized interface
    - Progressive Web App (PWA)

---

## 📚 Learning Resources

### Technologies Covered

- Python web development
- JavaScript React
- REST APIs
- Database design
- Cloud API integration
- Environmental monitoring

### Skills Gained

- Full-stack development
- System design

- API integration
- Data analysis
- Sustainability awareness

---

## 🎯 Key Takeaways

☑ **Green Model Advisor combines:**

- Modern web technologies
- AI/ML services
- Real-time monitoring
- Environmental consciousness

☑ **It demonstrates:**

- Full-stack architecture
- Multi-service integration
- Practical AI application
- Sustainability in tech

☑ **Perfect for:**

- Learning full-stack development
- Understanding AI models
- Exploring green computing
- Building production systems

---

## 📞 Questions About Technologies?

**Want to know more about a specific technology?**

- Check the main README.md for setup
- Look at source code comments
- Review the API documentation at `/docs`
- Check technology official websites:
  - React: react.dev
  - FastAPI: fastapi.tiangolo.com
  - CodeCarbon: codecarbon.io
  - SQLAlchemy: sqlalchemy.org

---

**Made with 🌍 to help you understand modern AI technology**