🌅 Morning (First 1-2 Hours): Understand the Problem Spec

Step 1: Read and Digest the Problem (15-20 mins)

- Carefully read the problem description **twice**.
- Highlight **key requirements**:
    - Input/output formats.
    - Constraints (e.g., performance limits, scale).
    - Metrics (accuracy, latency, memory usage).

Step 2: Ask Clarifying Questions (10-15 mins)

- Are examples/data formats given? If not, **ask for samples**.
- Clarify edge cases and assumptions (e.g., missing values, empty inputs).
- Confirm if specific libraries (e.g., **Pandas**, **PyTorch**, **FastAPI**) are **allowed or required**.

Step 3: Brainstorm the Solution (30-45 mins)

- **Break the problem into smaller steps**:
    1. Data ingestion and preprocessing.
    2. Model building/training (if ML is required).
    3. Backend API setup for deployment.
    4. Monitoring, logging, scalability (if needed).
- Draw **flowcharts** or **diagrams** for pipelines and data flow (use tools like **Excalidraw** or **Miro**).
- Write **pseudo-code** to outline your approach.

Deliverable:

- Finalize the **design and architecture**.
- Identify libraries/tools required.
- Decide on the **data structure** for inputs and outputs.

🌞 Mid-Morning (2-3 Hours): Build the Core Solution

Step 4: Start with the Model or Core Algorithm (1.5–2 hours)

- **Focus on the backbone first**:
    - Write a **data loader** (handle CSV, JSON, etc.).
    - Preprocess inputs (cleaning, scaling, embeddings).
    - Build the **model or algorithm** using **scikit-learn**, **PyTorch**, or **TensorFlow**.

    - Train/test the model (or implement logic) locally with sample data.

Test with small batches:
python
Copy code
```
assert model.predict(sample_input) == expected_output
```

-
- **Optimize early**:
    - Use batching, caching, and vectorization.
    - Optimize pipelines for speed.

Step 5: Build the Backend API (1 hour)

Set up a **FastAPI** or **Flask** server:
bash
Copy code
```
pip install fastapi uvicorn
```

-
- **Focus on endpoints**:
    - POST `/predict`: Accepts inputs and returns predictions.
    - POST `/train`: Retrains the model if required.

Example API Skeleton:

python
Copy code
```
from fastapi import FastAPI
import pickle
import pandas as pd

app = FastAPI()

@app.post("/predict/")
async def predict(data: dict):
    model = pickle.load(open("model.pkl", "rb"))
    df = pd.DataFrame([data])
    prediction = model.predict(df)
    return {"prediction": prediction.tolist()}
```

Test locally using:
bash
Copy code
```
uvicorn main:app --reload
```

-

Test with Postman or cURL:
bash
Copy code
```
curl -X POST "http://127.0.0.1:8000/predict/" -H "Content-Type: application/json" -d '{"x1": 0.5, "x2": 1.2}'
```

-

🌅 Afternoon (3-4 Hours): Make It Robust

Step 6: Handle Edge Cases (1 Hour)

- Test **corner cases**:
  - Missing/invalid data.
  - Empty inputs or out-of-range values.
  - High-throughput requests.

Example Validation:

python
Copy code
```
from pydantic import BaseModel

class InputSchema(BaseModel):
    feature1: float
    feature2: float
```

Step 7: Optimize and Refactor (1 Hour)

- **Optimize ML Model**:

Hyperparameter tuning with **Optuna**:
python
Copy code
```
from optuna import create_study
```

  -
  - Reduce inference time (e.g., **ONNX Runtime** or **TorchScript**).
  - Use **FP16** precision for embeddings (if ML-heavy).
- **Code Quality**:
  - Remove duplicates or redundant loops.
  - Split into **helper functions**.

Step 8: Add Logging and Monitoring (30-45 mins)

- Use **logging**:

python
Copy code
```
import logging
logging.basicConfig(level=logging.INFO)
```

- Monitor latency:

python
Copy code
```
start = time.time()
prediction = model.predict(inputs)
end = time.time()
logging.info(f"Inference Time: {end - start}s")
```

- Add basic **metrics**:

python
Copy code
```
from prometheus_client import start_http_server, Counter

REQUEST_COUNT = Counter('app_requests_total', 'Total API requests')
```

🌛 Evening (Final 1-2 Hours): Testing and Deployment

Step 9: Write Unit Tests (30-45 mins)

- Use **Pytest** to write tests:

bash
Copy code
```
pip install pytest
```

Example Test:

python
Copy code
```
from fastapi.testclient import TestClient
from main import app
```

```
client = TestClient(app)

def test_predict():
    response = client.post("/predict/", json={"feature1": 0.5,
"feature2": 1.2})
    assert response.status_code == 200
    assert "prediction" in response.json()
```

Run Tests:

bash
Copy code
```
pytest
```

Step 10: Dockerize (30 mins)

Dockerfile:

dockerfile
Copy code
```
FROM python:3.9
WORKDIR /app
COPY . /app
RUN pip install -r requirements.txt
CMD ["uvicorn", "main:app", "--host", "0.0.0.0", "--port", "8000"]
```

Build and Run Container:

bash
Copy code
```
docker build -t ml-app .
docker run -p 8000:8000 ml-app
```

Step 11: Final Checks (30 mins)

- Test API with **Postman** or **cURL** for edge cases.
- Stress test with **locust** or **k6**:

bash

Copy code
```
pip install locust
```

python
Copy code
```
from locust import HttpUser, task

class MLApiUser(HttpUser):
    @task
    def predict(self):
        self.client.post("/predict", json={"feature1": 0.5,
"feature2": 1.2})
```

Run Load Test:

bash
Copy code
```
locust -f locustfile.py
```

Final Strategy Recap

Time Block	Task	Goal
Morning (1-2 hours)	Understand the problem, brainstorm design.	Clear requirements and architecture.
Mid-Morning (2-3 hours)	Build the core ML model and backend API.	Working backbone, model outputs.
Afternoon (3-4 hours)	Optimize, handle edge cases, add logs.	Stability and robustness.
Evening (1-2 hours)	Write tests, Dockerize, and stress test the API.	Production-ready, deployable solution.

Key Tips for Success

1. **Write First, Optimize Later**:
    - Get a working prototype first, then optimize step-by-step.
2. **Focus on Critical Features**:
    - Don’t over-engineer—stick to the requirements.
3. **Log Everything**:

    - Track errors, latency, and predictions for debugging.
4. **Test Early**:
    - Write simple tests as you build each component.
5. **Keep It Modular**:
    - Use classes and helper functions to simplify the design.
6. **Stay Calm and Iterate**:
    - If stuck, **Google strategically** and reuse existing patterns or libraries.

You've got this! 🏆 Let me know if you need more examples or breakdowns during prep! 🚀

You said:

INTERLUDE

Ah! Now I get it! 🚀

You're asking how to efficiently decide where to search for solutions, templates, and examples—so you don't waste time digging aimlessly during the coding challenge.

Here's the Ultimate Search Strategy for Coding Problems 🔍:
1. Problem Classification (60–90 seconds)
Before you even search, categorize the issue:

Conceptual Problem – Do I need theory or explanations?
Code Implementation – Do I need working examples or templates?
Debugging Issue – Is it a specific error in syntax, imports, or APIs?
Scalability Concern – Am I optimizing for speed, memory, or deployment?
2. Search Sources by Problem Type
A. Algorithms, Math, Theory (Concepts)
Google Scholar → For mathematical models, loss functions.
Wikipedia → Quick formulas and definitions.
Towards Data Science (Medium) → Intuitive ML explanations with code.
Papers with Code → Latest research + implementations.
GitHub Repos → Find notebooks with step-by-step examples.
Search Examples:

"Binary Cross Entropy formula explanation site:medium.com"
"Transformer architecture code implementation site:paperswithcode.com"
B. Framework/API-Specific Syntax (PyTorch, FastAPI, etc.)
Official Docs → First stop for API-level issues.
HuggingFace Documentation → Pre-trained models (e.g., BERT, SBERT).
PyTorch Forums → Deep-dive errors in tensors, CUDA, etc.
TensorFlow Hub → Tensor-related problems if using TF/Keras.
FastAPI Docs → Templates for backend APIs.

Search Examples:

"PyTorch DataLoader batch_size error site:pytorch.org"
"FastAPI endpoint request body example site:fastapi.tiangolo.com"
C. Error Messages (Debugging)
Stack Overflow → #1 for debugging—focus on error codes.
GitHub Issues → If it's library-specific, check related issues.
Reddit (r/learnpython) → Good for obscure errors and workarounds.
Google Search → Include error text in quotes for exact matches.
Search Examples:

"RuntimeError: CUDA out of memory site:stackoverflow.com"
"torch.save model not working site:github.com"
D. Design Patterns and Architecture Templates
Awesome Lists on GitHub → Curated templates for ML, MLOps, APIs.
Repo: Awesome Machine Learning
MLFlow & Prefect Templates → MLOps examples with tracking and pipelines.
Full Stack FastAPI Template → GitHub Repo
Search Examples:

"Awesome FastAPI templates GitHub"
"MLOps pipeline examples with Prefect GitHub"
E. Performance Optimization (Speed/Memory)
HuggingFace Forums → Quantization, pruning, and model distillation.
PyTorch Forums → Multi-threading, batching optimizations.
Medium/Towards Data Science → Real-world optimization examples.
Search Examples:

"PyTorch inference optimization batch_size site:discuss.pytorch.org"
"Quantize BERT for faster inference site:huggingface.co"
F. Deployment, DevOps, Scaling
FastAPI and Flask Docs → Deployment configs, Docker setups.
Kubernetes Documentation → YAML templates for clusters.
AWS/GCP Docs → Deployment pipelines.
Reddit (r/devops) → Practical scaling tips.
Google Cloud Run Examples → Serverless templates.
Search Examples:

"Deploy FastAPI with Docker site:docs.docker.com"
"Horizontal Pod Autoscaling Kubernetes YAML template site:kubernetes.io"
G. Testing and Validation
Pytest Docs → Unit and functional test examples.
Locust Docs → Load testing for APIs.
Postman Blog → Manual and automated API testing.
Search Examples:

"pytest example for FastAPI endpoints site:docs.pytest.org"
"Locust performance testing example site:locust.io"

H. Ready-to-Use Templates and Boilerplates
GitHub Templates:
Machine Learning Boilerplates
FastAPI + ML
Kaggle Notebooks → Fully functional ML pipelines.
HuggingFace Pipelines → Pre-built models for tasks.
AI Template Repos → Airflow Templates
Search Examples:

"FastAPI Docker ML pipeline GitHub template"
"Pinecone + SBERT search GitHub repo"
3. Search Modifiers for Faster Results
A. Target Specific Sites:
site:stackoverflow.com → Focus on Stack Overflow.
site:github.com → Search only repositories.
site:pytorch.org → Target PyTorch documentation.
site:medium.com → Find tutorials and blog posts.
Example:

bash
Copy code
SBERT cosine similarity example site:pytorch.org
B. Focus on File Types:
Templates: filetype:ipynb → Jupyter Notebooks.
Diagrams: filetype:pdf → Research papers.
Example:

bash
Copy code
ML pipeline design pdf filetype:pdf
C. Combine Keywords:
Add error codes or functions in quotes for exact matches:
bash
Copy code
"RuntimeError: CUDA out of memory" site:stackoverflow.com
4. Pro Tips During the Test
Bookmark Key Pages:

Keep docs open: PyTorch, FastAPI, HuggingFace, Docker, Kubernetes.
Template Library:

Save GitHub repos for ML and backend templates before the test.
Manage Time:

Search Time Limit → No more than 10 minutes per problem.
Fallback: Start coding even with partial understanding—optimize later.
Ask ChatGPT for Syntax:

Use ChatGPT or Copilot to generate quick templates/snippets when you're stuck.

Final Example Search Map

Need	Search Query Example	Go-to Source
PyTorch DataLoader batching error	DataLoader batch size error site:discuss.pytorch.org	PyTorch Forums
FastAPI endpoint with POST request body	FastAPI post endpoint example site:fastapi.tiangolo.com	FastAPI Docs
Deploy Docker container with FastAPI	Dockerfile example for FastAPI ML site:github.com	GitHub Templates
Quantize BERT for faster inference	quantize bert inference site:huggingface.co	HuggingFace Docs and Forums
ML pipeline YAML for Kubernetes	kubernetes ml pipeline template site:kubernetes.io	Kubernetes Docs + GitHub Examples