APIs & Model Demos — Follow-Along Guide

Week 12 · CS 203 · Software Tools and Techniques for AI

Prof. Nipun Batra · IIT Gandhinagar

Spring 2026

How to Use This Guide

- Open `apis_followalong.sh` in your editor (left half of screen)
- Open a terminal (right half of screen)
- Copy-paste each command, one at a time
- **Type it yourself** — that's how you learn

Legend: `$` = command to type. `>>` = expected output. Blue boxes = projector slide.

Projector: Slides 2–3 — From Notebook to Product Your model lives in a notebook. Nobody can use it. Today we fix that: API → demo → Docker.

Act 1: Train and Save a Model ~8 min

```
$ mkdir -p ~/api-demo && cd ~/api-demo
$ python -m venv .venv && source .venv/bin/activate
$ pip install scikit-learn numpy fastapi uvicorn joblib gradio streamlit
$ python train.py
>> Model accuracy: 0.xxx
>> Model saved to model.pkl
```

Tip

`joblib.dump(model, "model.pkl")` saves the trained model. `joblib.load("model.pkl")` loads it back. Load once at server startup, not per request.

Projector: Slides 4–8 — REST Theory HTTP methods: GET (read), POST (send). Status codes: 200 (ok), 422 (bad input), 500 (error). Stateless = no memory between requests.

Act 2: FastAPI — Your First API ~15 min

Create `app.py` with FastAPI:

```
@app.post("/predict", response_model=Prediction)
def predict(features: MovieFeatures):
    X = np.array([[features.budget, ...]])
    pred = model.predict(X)[0]
    proba = model.predict_proba(X)[0]
    return Prediction(success=bool(pred), ...)
```

Start and test:

```
$ uvicorn app:app --reload --port 8000 &
$ curl -s http://localhost:8000/ | python -m json.tool
>> {"status": "healthy", "model": "random_forest_v1"}
$ curl -s -X POST http://localhost:8000/predict \
    -H "Content-Type: application/json" \
    -d '{"budget": 0.8, "runtime": 0.5, "genre_action": 1, ...}'
>> {"success": true, "confidence": 0.85, "label": "Hit!"}
```

Open `http://localhost:8000/docs` — interactive Swagger UI, auto-generated!

Act 3: Testing Your API ~5 min

```
$ pytest test_api.py -v
>> test_api.py::test_health PASSED
>> test_api.py::test_predict PASSED
>> test_api.py::test_predict_invalid PASSED
```

Warning

Always test the 422 case — send invalid input and verify the API rejects it. Pydantic handles validation automatically with FastAPI.

Projector: Slides 13–15 — Gradio and Streamlit Gradio = instant ML demo in 10 lines. Streamlit = richer dashboard.

Act 4: Gradio — Demo in 10 Lines ~10 min

```
demo = gr.Interface(
    fn=predict,
    inputs=[gr.Slider(0, 1, label="Budget"), ...],
    outputs=gr.Text(label="Prediction"),
)
demo.launch()
```

```
$ python app_gradio.py
>> Running on http://127.0.0.1:7860
```

Tip

Add `share=True` to `demo.launch()` to get a public URL — great for sharing demos with non-technical people.

Act 5: Streamlit — Dashboard Demo ~10 min

```
$ streamlit run app_streamlit.py
>> Opens at http://localhost:8501
```

Streamlit reruns the entire script on each interaction. Gradio calls your function on submit.

Act 6: Dockerize the API ~10 min

```
$ docker build -t movie-api .
$ docker run -d -p 8000:8000 --name movie-api movie-api
$ curl -s http://localhost:8000/predict -X POST \
    -H "Content-Type: application/json" \
    -d '{"budget": 0.8, ...}'
>> Same result --- runs identically in Docker!
```

Quick Reference

I want to…	Command
Start FastAPI server	`uvicorn app:app --reload`
View auto-docs	Open `http://localhost:8000/docs`
Test with curl	`curl -X POST url -H "Content-Type: application/json" -d '{...}'`
Run Gradio demo	`python app_gradio.py`
Run Streamlit	`streamlit run app.py`
Save sklearn model	`joblib.dump(model, "model.pkl")`
Load sklearn model	`model = joblib.load("model.pkl")`
Build Docker image	`docker build -t name .`
Run container	`docker run -p 8000:8000 name`

Exam-relevant concepts:

- HTTP methods: GET (read), POST (create/predict), PUT (update), DELETE (remove)
- Status codes: 200 (OK), 400 (bad request), 404 (not found), 422 (validation), 500 (server error)
- Online inference: one prediction, low latency ($<$100ms)
- Batch inference: many predictions, latency doesn't matter
- Stateless services: no memory between requests → easy to scale
- Model serialization: joblib (sklearn), pickle, torch.save (PyTorch)