Course Summary & What's Next

Week 12: CS 203 - Software Tools and Techniques for AI

Prof. Nipun Batra

IIT Gandhinagar

The Journey

In January, you knew how to write a Python script.

Now you can:

- **Collect** data from APIs, web pages, and sensors
- **Validate** it with schemas before it poisons your model
- **Label** it efficiently (and know when to use LLMs to help)
- **Augment** it when you don't have enough
- **Evaluate** models properly (not just accuracy!)
- **Tune** hyperparameters (and not waste time doing it)
- **Track** experiments so you never lose a good run
- **Dockerize** your work so anyone can reproduce it
- **Deploy** it as a web app anyone can use
- **Profile** and **optimize** it for real-world constraints

The Full Stack, One Slide

```
Raw Data
  │
  ├── Collect (APIs, scraping, sensors)     ← Week 1-2
  ├── Validate (schemas, types, ranges)     ← Week 2
  ├── Label (manual, active, weak, LLMs)    ← Week 3-4
  ├── Augment (transforms, synthetic)       ← Week 5
  │
  ├── Train (sklearn, pipelines)
  ├── Evaluate (CV, bias-variance)          ← Week 7
  ├── Tune (grid, random, Bayesian)         ← Week 8
  ├── Track (TrackIO, seeds)                ← Week 9
  │
  ├── Deploy (Flask, Streamlit, Gradio)     ← Week 10
  ├── Containerize (Docker, HF Spaces)      ← Week 9-10
  ├── Profile & Optimize (ONNX, INT8)       ← Week 11
  │
```

Week 1-2: Data Collection & Validation

Data Collection: What We Learned

APIs — structured data from services (JSON, REST)

```
response = requests.get("https://api.example.com/data")
data = response.json()
```

Web Scraping — extracting data from HTML pages

```
from bs4 import BeautifulSoup
soup = BeautifulSoup(html, "html.parser")
titles = soup.find_all("h2")
```

Key insight: The data you collect determines your model's ceiling. No algorithm can fix bad data.

Data Validation: What We Learned

Pydantic — validate data with Python type hints

```
class SensorReading(BaseModel):
    temperature: float = Field(ge=-50, le=60)
    humidity: float = Field(ge=0, le=100)
    timestamp: datetime
```

Key insight: Validate at the boundary. Catch bad data before it enters your pipeline, not after your model produces garbage predictions.

Week 3-5: Data Labeling & Augmentation

Labeling: The Bottleneck

Strategy	When to Use
Manual labeling	Small datasets, high-stakes domains
Active learning	Large unlabeled pool, expensive labels
Weak supervision	Heuristics + noisy labels at scale
LLM-assisted	Text tasks, when budget allows API calls

Key insight: Labeling is usually the most expensive part of ML. The right strategy can save weeks of work.

Augmentation: More Data from Existing Data

Domain	Techniques
Images	Flip, rotate, crop, color jitter
Text	Synonym replacement, back-translation, paraphrase
Tabular	SMOTE, noise injection

Key insight: Augmentation is free data. But it must be *semantically valid* — flipping a 6 to make a 9 doesn't help a digit classifier.

Week 6-8: LLMs, Evaluation & Tuning

LLM APIs: What We Learned

```
response = client.messages.create(
    model="claude-sonnet-4-20250514",
    messages=[{"role": "user", "content": prompt}]
)
```

Key insight: LLMs are tools, not magic. Prompt engineering, structured outputs, and knowing when NOT to use an LLM are the real skills.

Evaluation: Beyond Accuracy

Concept	Why It Matters
Cross-validation	One train/test split is a coin flip
Stratified K-Fold	Preserves class balance in each fold
Bias-variance tradeoff	Underfitting vs overfitting, visualized
Leakage	The #1 cause of "too good to be true" results

Key insight: If you only do one thing, do **5-fold cross-validation** instead of a single train/test split. It's the minimum viable evaluation.

Tuning: Don't Waste Time

Method	Tries	Finds Best?	When to Use
Grid Search	All combos	Eventually	< 3 hyperparameters
Random Search	Random subset	Often faster	3+ hyperparameters
Bayesian (Optuna)	Smart picks	Yes	Expensive evaluations

Key insight: Random search beats grid search in almost all cases. Don't tune what doesn't matter.

Week 9-10: Reproducibility & Deployment

Reproducibility: Three Levels

Level	Problem	Solution
The Math	Different results every run	`random_state=42`
The Memory	Lost the best run	TrackIO
The Machine	Works on my laptop only	Docker

```
# Seeds
X_train, X_test = train_test_split(X, y, random_state=42)

# Tracking
trackio.init(project="my-project", config={...})
trackio.log({"accuracy": 0.95})
trackio.finish()

# Docker
# docker build -t my-app . && docker run -p 7860:7860 my-app
```

Deployment: Four Frameworks

Framework	Best For	Lines of Code
Flask	Custom web apps (you write HTML)	~40
Streamlit	Data dashboards (pure Python)	~15
Gradio	ML demos (define inputs/outputs)	~10
FastAPI	Production APIs (JSON in/out)	~20

Key insight: Use Gradio for demos, FastAPI for production, HuggingFace Spaces for free hosting.

Week 11: Profiling & Optimization

Profiling & Quantization: What We Learned

Profiling: Find the bottleneck before optimizing.

```
time.time() → cProfile → line_profiler → memory_profiler
```

Quantization: Make models smaller with minimal accuracy loss.

```
FP32 (4 bytes) → INT8 (1 byte) = 4x smaller, < 1% accuracy drop
```

ONNX: Train in Python, run anywhere (mobile, browser, edge).

Key insight: The #1 speedup for web apps is loading the model once at startup instead of per-request. Profile before you optimize.

Connecting the Dots

The Tools Map

PLAN	BUILD	SHIP	OPTIMIZE
Validation	sklearn	Flask	cProfile
Schemas	Pipelines	Streamlit	line_profiler
Labeling	CV/Tuning	Gradio	Quantization
Augmentation	TrackIO	FastAPI	ONNX
	Seeds	Docker	
		HF Spaces	

You don't need all of these for every project. But you need to know they exist so you can pick the right tool for the job.

The Minimum Viable ML Project

For your course project, this is the minimum:

```
my-project/
├── data/                  # raw + processed data
├── notebooks/             # exploration
├── src/
│   ├── train.py          # training script with random_state
│   └── app.py            # Gradio/Streamlit app
├── requirements.txt       # pinned versions
├── Dockerfile             # optional but impressive
└── README.md              # how to run it
```

With:

- Seeds set everywhere
- At least 5-fold CV
- TrackIO logging
- A working demo someone can actually run

What This Course Didn't Cover (But You Should Know Exists)

Topic	Why It Matters	Where to Learn
Deep Learning	CNNs, transformers, fine-tuning	CS 337, fast.ai
MLOps	CI/CD, model registries, monitoring	Made With ML
Cloud Deployment	AWS/GCP/Azure, Kubernetes	Cloud provider docs
Data Engineering	ETL pipelines, data warehouses	dbt, Airflow
ML System Design	Scaling, A/B testing, feedback loops	Chip Huyen's book

This course gave you the **foundations**. These are the next steps.

Advice for Your Future Projects

1. **Start with the data, not the model.** 80% of ML is data work.
2. **Simple models first.** Logistic regression before transformers. You'd be surprised how often simple wins.
3. **Automate the boring stuff.** Seeds, tracking, Docker, requirements.txt — set these up on day 1, not day last.
4. **Make it runnable.** If someone can't clone your repo and run your code in 5 minutes, it doesn't exist.
5. **Read error messages.** The answer is almost always in the traceback. Read it before opening Stack Overflow.

Thank You

This was CS 203: **Software Tools and Techniques for AI**.

You started with scripts. You end with deployable, reproducible, optimized ML systems.

Go build something.

```
Seeds  →  TrackIO  →  Docker  →  Gradio  →  HF Spaces
  ↓         ↓          ↓          ↓          ↓
Control   Remember   Reproduce   Share      Deploy
```