Course Summary & What's Next

Week 13: CS 203 — Software Tools and Techniques for AI

Prof. Nipun Batra

IIT Gandhinagar · Spring 2026

The Journey — Data Side

In January, you could write a Python script. Now you can:

- **Collect** data from APIs, web pages, sensors
- **Validate** it with schemas before it poisons your model
- **Label** it efficiently (manual, active, weak, LLM-assisted)
- **Augment** it when you don't have enough

The Journey — Models & Production

- **Call LLMs** with structured outputs, multimodal inputs
- **Evaluate** properly — not just accuracy
- **Tune & track** hyperparameters
- **Reproduce** runs across machines (Docker, seeds)
- **Monitor** drift in production
- **Profile & quantize** for real-world constraints
- **Build agents** that use tools to take actions

One-Liner Per Lecture — Data

#	Week	Takeaway
1	Data Collection	*Paginate, retry, validate status codes.*
2	Data Validation	*Validate at the boundary — loud errors, not silent bugs.*
3	Data Labeling	*Annotation guidelines matter more than the tool.*
4	Optimizing Labeling	*Active + weak supervision turn labels into code.*
5	Data Augmentation	*Free data — if the transform preserves the label.*

One-Liner Per Lecture — Models

#	Week	Takeaway
6	LLM APIs	*Prompts are functions; version and test them.*
7	Model Evaluation	*Single split = coin flip; 5-fold CV is the floor.*
8	Tuning + Tracking	*Random > grid past 3 HPs; Optuna > random when evals are expensive.*

One-Liner Per Lecture — Production

#	Week	Takeaway
9	Reproducibility	*Fix the machine, the math, and the memory — all three.*
10	Data Drift	*Monitor distributions, not just metrics.*
11	Profiling + Quantization	*Profile first; INT8 is ~4× free compression.*
12	AI Agents	*LLM + tools + while-loop = Claude Code, Cursor, Devin.*
—	Web Apps / CI/CD / APIs	*If it doesn't run in 5 minutes, it doesn't exist.*

The Full Stack

```
Raw Data
  │
  ├── Collect    (APIs, scraping)            ← Week 1-2
  ├── Validate   (schemas, types, ranges)    ← Week 2
  ├── Label      (manual, active, weak, LLM) ← Week 3-4
  ├── Augment    (transforms, synthetic)     ← Week 5
  ├── LLM APIs   (Gemini, prompts)           ← Week 6
  ├── Evaluate   (CV, bias-variance)         ← Week 7
  ├── Tune/Track (Optuna, TrackIO)           ← Week 8
  ├── Reproduce  (Docker, seeds)             ← Week 9
  ├── Monitor    (drift detection)           ← Week 10
  ├── Profile    (INT8, ONNX, pruning)       ← Week 11
  └── Agent      (tools, loops)              ← Week 12
```

Each layer depends on the ones above. Skip one, ship a bug.

Part I — Data (Weeks 1–5)

"80% of ML is data work."

Data Collection — APIs

```
r = requests.get("https://api.example.com/data", timeout=5)
r.raise_for_status()     # crash loudly, don't silently corrupt
data = r.json()
```

Key insight: the data you collect sets your model's *ceiling*.
No algorithm fixes bad data.

Data Collection — Scraping & Gotchas

```
from bs4 import BeautifulSoup
soup = BeautifulSoup(html, "html.parser")
titles = [h.get_text().strip() for h in soup.find_all("h2")]
```

Gotchas you met this semester:

- Rate limits & backoff
- Pagination cursors
- JS-rendered pages (needs Playwright/Selenium)
- Character encoding (`utf-8` vs `latin-1`)
- Time-zone bugs

Data Validation — Pydantic

```
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.

Labeling — Strategies

Strategy	When to use	Cost / sample
Manual	Small, high-stakes	Highest
Active learning	Large unlabeled pool	Medium
Weak supervision	Domain heuristics	Low
LLM-assisted	Text tasks, budget for API	Low but ≠ free

Labeling is usually the **most expensive part of ML**.

Active Learning in One Line

$$\text{next\_to\_label} = \arg\max_{x \in \mathcal{U}} H\big(p(y \mid x)\big)$$

Pick the **most uncertain** unlabeled example.

- Ties the model's own confidence to the labeling budget
- Typically 5–10× fewer labels for the same accuracy
- Entropy is the standard uncertainty measure; margin / disagreement also work

Weak Supervision in 10 Lines

```
def lf_positive(text):
    return 1 if any(w in text for w in ["great", "excellent"]) else -1

def lf_negative(text):
    return 0 if any(w in text for w in ["bad", "terrible"]) else -1

labels = [lf_positive(t), lf_negative(t)]
noisy_label = Counter(l for l in labels if l != -1).most_common(1)[0][0]
```

Key insight: 70–90% of manual-labeling accuracy from hand-written rules + a denoiser. That's the Snorkel playbook.

Augmentation — Free Data *(if label-preserving)*

Domain	Techniques
Images	flip, rotate, crop, color jitter, RandAugment
Text	synonym replacement, back-translation
Audio	time shift, pitch shift, noise injection
Tabular	SMOTE, noise injection, mixup

⚠️

Canonical mistake: flipping a handwritten `6` into a `9`.

Always ask: *does this transform preserve the label?*

Part II — Models (Weeks 6–8)

"A simple model you can trust beats a complex one you can't."

LLM APIs

```
from google import genai
client = genai.Client()

response = client.models.generate_content(
    model="gemini-2.0-flash",
    contents=prompt,
    config={"response_mime_type": "application/json",
            "response_schema": ReviewSentiment},
)
```

Four skills: structured outputs · multimodal · streaming · prompt versioning.

When *Not* to Use an LLM

Task	Better choice
Exact arithmetic	A calculator
Date parsing	`datetime.strptime`
Structured extraction	regex / small fine-tuned model
Classification with lots of labels	logistic regression / BERT
Anything that must be deterministic	Pretty much anything else

LLMs are a tool, not a default.

Evaluation — Beyond Accuracy

Concept	Why it matters
Cross-validation	Single split is a coin flip
Stratified K-Fold	Preserves class balance
Nested CV	Needed when you tune hyperparameters
TimeSeriesSplit	Time-ordered data needs time-ordered folds
GroupKFold	Patients / users must not span folds

Minimum viable evaluation: **5-fold stratified CV**.

Bias–Variance in One Formula

$$\mathbb{E}[(y - \hat{f}(x))^2] = \underbrace{\left(\mathbb{E}[\hat{f}] - f\right)^2}_{\text{bias}^2} + \underbrace{\mathrm{Var}\left[\hat{f}\right]}_{\text{variance}} + \sigma^2$$

- Simple models → high bias, low variance (underfit)
- Complex models → low bias, high variance (overfit)
- Pick where **test** error bottoms out, not training error

Data Leakage — The #1 Killer

Test-time information sneaking into training:

- **Scaling on the full dataset before splitting**
- Using a target-derived feature (`log_of_price` to predict `price`)
- Time-series data split randomly instead of chronologically
- Duplicated rows across train / test
- Test set peeking during hyperparameter tuning

Rule of thumb: 99% where everyone else is at 85% → you have a leak.

Tuning — Don't Waste Time

Method	Tries	When
Grid Search	all combos	< 3 hyperparams
Random Search	random subset	3+ hyperparams
Bayesian (Optuna)	adaptive	expensive evals
Hyperband / ASHA	early stopping	deep learning

Bergstra & Bengio 2012: only ~2 of your 5 hyperparams matter, and random explores those coordinates more densely than grid.

Experiment Tracking in 4 Lines

```python
import trackio
run = trackio.init(project="moons", config={"lr": 1e-3, "hidden": 16})
for epoch in range(100):
    trackio.log({"loss": loss.item(), "acc": acc.item()})
run.finish()
```

Answers *"which run was the good one?"* — without this you're re-running experiments by hand.

Part III — Production (Weeks 9–11)

"A model in a notebook is not a model."

Reproducibility — Three Layers

Layer	Symptom	Cure
The Math	Different results each run	seeds everywhere
The Memory	Lost the best hyperparams	experiment tracking
The Machine	"Works on my laptop"	Docker / `pyproject.toml`

All three matter. Missing one is a reproducibility bug.

Docker — The 30-Second Explanation

```
FROM python:3.11-slim
WORKDIR /app
COPY requirements.txt .
RUN pip install -r requirements.txt
COPY . .
CMD ["python", "app.py"]
```

- `docker build` turns your code into a portable image
- `docker run -p 7860:7860` ships to any Linux / Mac / CI server
- No more *"install these 14 things first"*

Data Drift — Tests

Test	Detects
KS test	Shift in continuous features
PSI	Population stability (train vs prod)
Chi-squared	Categorical feature shifts
Jensen–Shannon / Wasserstein	General distribution distance

Data Drift — KS Statistic

$$\mathrm{KS} = \max_x \left| F_{\text{train}}(x) - F_{\text{prod}}(x) \right|$$

- F = empirical CDF of each distribution
- Big KS ⇒ distributions drifted apart
- Pair with a p-value from the two-sample KS test

Key insight: 95% accuracy at deployment → silent 70% two months later if the input distribution shifts.

Profiling — Find Before You Fix

```
print(time.time() - t0)   # first check: how slow?
cProfile                  # which function?
line_profiler             # which line?
memory_profiler           # which allocation?
torch.profiler            # GPU timing & kernel breakdown
```

Key insight: the #1 speedup in student projects this semester was loading the model **once at startup** instead of per request.

Quantization in Three Lines

$$\text{scale} = \frac{\max|W|}{127}, \quad q = \text{round}\big(W/\text{scale}\big), \quad \hat{W} = q \cdot \text{scale}$$

```python
def quantize_tensor(x):
    s = x.abs().max().item() / 127.0
    q = torch.round(x / s).clamp(-127, 127).to(torch.int8)
    return q, s
```

4× smaller weights, typically **< 1%** accuracy drop.

Quantization Scales — Toy to 0.5B LLM

Same 3-line routine applied to progressively bigger models:

Notebook	Model	Params	INT8 vs FP32
`09-…from-scratch`	MLP · `make_moons`	~400	≈ 0
`11-…llm-from-scratch`	2-layer Transformer · Hamlet	~60K	Δ loss ≈ 0.005
`12-…real-llm`	**Qwen 2.5 0.5B** (HF)	**494M**	< 1% CE

The math doesn't care about model size.

Beyond Quantization — Compression Toolbox

Technique	Idea	Typical gain
Quantization	Fewer bits per weight	4× (INT8), 8× (INT4)
Pruning	Zero out small weights	50–90% sparsity
Distillation	Small "student" mimics big "teacher"	5–10× smaller model
ONNX export	Portable graph format	Runs anywhere (mobile, browser, edge)

Real deployments stack these: distill → prune → quantize → ONNX.

Ship It — Demo Apps

```
# Gradio
import gradio as gr
gr.Interface(predict, "text", "label").launch()

# Streamlit
import streamlit as st
x = st.text_input("Prompt"); st.write(model(x))

# FastAPI
@app.post("/predict")
def predict(body: Input) -> Output: ...
```

Gradio for quick ML demos · Streamlit for dashboards · FastAPI for production APIs.

CI / CD in One Slide

```yaml
# .github/workflows/ci.yml
on: [push]
jobs:
  test:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v4
      - run: pip install -r requirements.txt
      - run: pytest
      - run: docker build -t app .
```

Every commit: install, test, build a Docker image. **If it breaks, you find out in 30 seconds, not two weeks later.**

Part IV — Agents (Week 12)

"The same pattern that builds Claude Code builds yours."

Agents — The Architecture

$$\textbf{agent} = \textbf{LLM} + \textbf{tools} + \textbf{loop}$$

Piece	Role	Analogy
LLM	Think, reason, decide	Brain
Tools	Functions the LLM can call	Hands
Loop	Keep going until done	Work ethic

You built a 4-tool agent with Gemma 4 on free Colab in ~100 lines.

Tool Calling — Minimum Viable Loop

```
TOOLS = [{"type": "function",
          "function": {"name": "calculate",
                       "description": "Do math",
                       "parameters": {...}}}]

while not done:
    response = llm(messages, tools=TOOLS)
    for call in response.tool_calls:
        result = dispatch(call)
        messages.append({"role": "tool", "content": result})
```

Bigger tools, same architecture. Claude Code, Cursor, Devin, Perplexity — all follow this shape.

Advanced Calculator — Why It's Not a Toy

The lab exercise: upgrade `calculate` to handle `sqrt, sin, log, factorial, np.mean, np.dot, matmul`, and keep `eval()` safe.

- **Exposes** that "tool = Python function" scales to arbitrary math
- **Teaches** sandboxing (`{"__builtins__": {}}`, AST walking)
- **Shows** that an LLM + numpy tool beats any LLM doing arithmetic alone

This is the real-world recipe: LLM + expert tool >> LLM alone.

Common Mistakes & Lessons

Top 10 Mistakes I Saw — Part 1

1. **Scaling before splitting** → data leakage → fake accuracy
2. **No seed set** → "my model worked yesterday"
3. **One train/test split** reported as final → coin flip
4. **Pickled the model without** `requirements.txt` → unreproducible
5. **Hardcoded paths** (`/home/student/...`) → breaks for the next person

Top 10 Mistakes I Saw — Part 2

6. **No README** → nobody can run it
7. **Test set peeking** during tuning → optimistic bias
8. **Committed API keys / secrets** → please read `.gitignore` docs
9. **Caught every exception broadly** → silent failures
10. **Optimized the wrong bottleneck** — profiling would have shown the fix in 30 s

Avoid these 10 → you're ahead of most production codebases.

Cheat Sheet — When Things Go Wrong

Cheat Sheet — Part 1

Symptom	Diagnosis	Lecture
"Accuracy changes every run"	No seed	9
"My script doesn't run for the TA"	Docker + pinned deps	9
"Accuracy dropped in production"	Data drift	10
"Test 99%, prod 70%"	Data leakage	7
"Model too slow"	Profile first	11

Cheat Sheet — Part 2

Symptom	Diagnosis	Lecture
"Model too big for device"	Quantize / prune / distill	11
"LLM hallucinates numbers"	Give it a calculator tool	12
"Lost my best hyperparams"	TrackIO / W&B	8
"Can't get enough labels"	Active / weak supervision	4
"Not enough data to train"	Augmentation	5

Screenshot these two slides. Come back in 5 years.

The Minimum Viable ML Project

```
my-project/
├── data/                  # raw + processed (gitignored)
├── notebooks/             # exploration only
├── src/
│   ├── train.py          # training with seeds + tracking
│   ├── evaluate.py       # reproducible metrics
│   └── app.py            # Gradio / Streamlit / FastAPI
├── tests/                 # at least one smoke test
├── requirements.txt       # pinned versions
├── Dockerfile             # recipes that always work
└── README.md              # runnable in < 5 min
```

Must have: seeds · 5-fold CV · tracking · a working demo.

What This Course Didn't Cover

Topic	Where to learn
Deep Learning (CNNs / transformers / fine-tuning)	CS 337, fast.ai
MLOps (registries, pipelines)	*Made With ML*
Cloud Deployment (AWS / GCP / Azure / k8s)	cloud docs
Data Engineering (ETL, warehouses)	`dbt`, Airflow
ML System Design (scaling, A/B, feedback)	Chip Huyen's book
MCP / A2A (agent tool standards)	Anthropic / Google docs

Foundations → yours. Next steps → above.

Advice for Future Projects

1. **Data before model.** 80% of ML is data work.
2. **Simple first.** Logistic regression before transformers.
3. **Automate day 1.** Seeds, tracking, Docker, `requirements.txt`.
4. **Make it runnable.** 5-minute clone → run, or it doesn't exist.
5. **Read the traceback** before asking ChatGPT.
6. **Version everything** — code, data, models, prompts.
7. **Ship early.** 70% demo today > 95% plan next month.

Monday Morning at a New ML Job

1. Find the `train.py`. Run it. Does it reproduce?
2. Ask: where is the source of truth for training data?
3. Add seeds if missing.
4. Add experiment tracking if missing.
5. Write a Dockerfile. Ship it even if imperfect.

You now know enough to be **the person** who does these things.

Thank You

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

You started with scripts. You end with reproducible, monitored, optimized ML systems — and agents that can take actions.

All slides, code, videos, labs:
https://nipunbatra.github.io/stt-ai-teaching/

Go build something.

Questions?

From scripts to systems. Go build.