# PHASE 8
# Capstone Projects

Compounding value by building end-to-end systems in your own domains

**Phase goal**

Convert knowledge into demonstrable capability. Ship 2–3 end-to-end projects that intersect your existing domains (logistics, bilingual documents, mobile, backend) so each new skill compounds with what you already do well.

**Estimated time:** Ongoing | **Track:** Comprehensive ML & Deep Learning Curriculum

A self-paced, depth-first path from mathematical foundations to production deep learning.

# Contents

# 1 The purpose of capstones

Everything before this phase built capability; capstones prove and compound it. A finished, deployed project that solves a real problem teaches more — and signals more — than another course completed. The strategic move is to pick projects at the *intersection* of machine learning and domains you already understand deeply. There your background is an unfair advantage: you can judge data quality, frame the problem correctly, and ship the production parts faster than a pure ML newcomer.

**Key idea**

Depth beats breadth here. Two or three *finished, deployed, documented* projects are worth more than ten notebooks that stop at "0.92 validation accuracy." For each capstone, the deliverable is a system someone could use, a repository someone could read, and a write-up that explains the decisions and the results honestly — including what did not work.

# 2 What "done" means for a capstone

| Dimension | Bar to clear |
|---|---|
| Problem framing | a real user/decision; a metric tied to value; a baseline |
| Data | sourced, cleaned, versioned; leakage checked |
| Modeling | a strong baseline first, then iteration justified by results |
| Evaluation | honest, with error bars and error analysis (not one number) |
| Deployment | served behind an API or app; reproducible; monitored |
| Communication | README + short write-up: problem, approach, results, limits |

**Intuition**

A reviewer (or future employer, or future you) should be able to clone the repo, read a one-page README, run one command to reproduce the core result, and hit a running endpoint. If they can, the project is "done." Polish the seams — they are what people actually see.

# 3 Recommended capstones (pick 2–3)

**1 · Shipping / logistics ML**

**Predictive delivery-time estimation or shipment anomaly detection** — ties directly into Atravesar / Canada Post work.

- *ETA regression:* predict delivery time from route, carrier, origin/destination, weight, season, and historical performance. Strong fit for gradient boosting (Phase 1); report MAE with calibrated prediction intervals.
- *Anomaly detection:* flag shipments likely to be lost/delayed/mis-scanned (isolation forest / autoencoder reconstruction error, Phases 1 & 5).
- *Production:* serve as an API your existing backend can call; monitor drift as carrier behavior and seasons change (Phase 7).

**Why it compounds:** you already understand the domain, the data sources, and the

integration surface — so you can focus on the ML and ship fast.

### 2 · NLP + your bilingual skill set

**Chinese↔English document intelligence** — leverage a genuinely rare combination.

- *Bilingual document classification* (e.g. routing or tagging mixed-language documents) with a multilingual Transformer (Phase 4).
- *Translation-quality estimation* or a *semantic search* tool over your own translated API docs.
- Mind the CJK tokenization quirks (Phase 4): measure token counts/costs and retrieval quality across languages.

**Why it compounds:** bilingual CJK/English ability plus ML is a differentiated niche; few engineers can build and *evaluate* this well.

### 3 · Mobile / on-device DL

**A lightweight on-device model for "Tips of Canada"** — e.g. receipt OCR + amount extraction.

- A small vision model for detection/recognition (Phase 3), converted to **CoreML** for iOS.
- Emphasize the on-device constraints: quantization, latency, size (Phase 7); compare cloud vs. on-device tradeoffs.

**Why it compounds:** exercises the full edge-deployment path and produces a tangible mobile demo.

### 4 · End-to-end RAG assistant

**A retrieval-augmented assistant over internal documentation** (e.g. SolvoSync docs).

- Full RAG pipeline (Phase 4): chunk → embed → vector store (pgvector fits an existing Postgres) → retrieve + rerank → generate with citations.
- Build a real evaluation harness (retrieval recall@k, answer faithfulness) and a simple chat UI or API.
- Deploy and monitor (Phase 7): track query patterns, latency, and answer quality.

**Why it compounds:** the highest-relevance modern skill, and a natural fit for your API/backend strengths.

### 5 · Kaggle for benchmarking

**Compete to calibrate yourself against the field.**

- Start with a "Getting Started" competition (Titanic) to learn the workflow, then enter a live competition for honest benchmarking.
- Study and reproduce top solutions afterward — reading winning write-ups teaches feature engineering and validation better than any course.

**Why it compounds:** forces rigorous validation and exposes you to techniques you would not invent alone.

## 4 Choosing your set

**Intuition**

A strong portfolio of three: **(2) the bilingual NLP project** (differentiated, modern), **(4) the RAG assistant** (highest current demand, plays to your backend), and **(1) the logistics project** (domain depth, clear business value). This trio spans classical ML, transformers, retrieval, and production deployment — exactly the compressed high-value path (Phases 0→1→2→4→7) made tangible.

## 5 Execution playbook

1. **Week 1 — frame & baseline.** Define the user, the metric, and a dumb baseline (majority class / simple heuristic / BM25 for RAG). Never skip the baseline; it is your reality check.
2. **Weeks 2–3 — iterate.** Improve data and model; every change justified by the validation metric. Keep an experiment log (Phase 7).
3. **Week 4 — productionize.** Serve, containerize, monitor, document. Resist endless modeling; shipping is the point.
4. **Throughout — write it down.** A running README and a final write-up (problem, data, approach, results with error bars, limitations, next steps).

**Common pitfall**

The two ways capstones die: **scope creep** (the project balloons until it is never finished) and **the 90% trap** (the model works in a notebook but is never deployed or documented). Fight both by fixing a deadline, defining "done" up front (the table in §2), and treating deployment and the write-up as first-class deliverables, not optional extras.

## 6 Staying current after the curriculum

**Keep the momentum**

- **Kaggle** — competitions and winning-solution write-ups.
- **Papers With Code** — papers paired with implementations; track SOTA on tasks you care about.
- **Hugging Face Papers / arXiv** — a steady, curated reading habit beats binge-then-forget.
- Re-read the reference books as references, not cover-to-cover: Géron, Huyen, Goodfellow et al.
- Build in public — a blog post or repo per capstone compounds reputation alongside skill.

## 7 Closing

You started at vectors and gradients and ended at deployed, monitored systems that learn. The throughline never changed: represent data with linear algebra, improve parameters with calculus, reason about uncertainty with probability, and engineer the system around it with care. From here, the field will keep moving — but the foundations you built will let every new paper, model, and tool read as a variation on themes you now own. Pick a capstone, set a deadline, and ship.

**Key idea**

Knowledge you can't apply fades; systems you've shipped stay with you. Make something real, write down how and why, and let each project make the next one easier. That compounding — not any single model — is the whole point of this curriculum.