ML System Design and Architecture

BECM33MLE — Machine Learning Engineering

Ing. Jan Brabec Ph.D.

CISCO

Lecture 4: ML System Design and Architecture

2025-10-16

ML System Design and Architecture
BECM33MLE — Machine Learning Engineering
Ing. Jan Brabec Ph.D.
CISCO

Ing. Jan Brabec Ph.D.

- Principal AI Researcher @ Cisco
- Building systems detecting cybersecurity threats
- Ph.D. on ML in Cybersecurity @ CTU Prague
- Thesis: "Class-Imbalanced Data in Cybersecurity"

Introduction

Ing. Jan Brabec Ph.D.
- Principal AI Researcher @ Cisco
- Building systems detecting cybersecurity threats
- Ph.D. on ML in Cybersecurity @ CTU Prague
- Thesis: "Class-Imbalanced Data in Cybersecurity"

"The [ML] algorithm is only a small part of an ML system in production."
— *Chip Huyen, Designing Machine Learning Systems*

Production ML is a multidisciplinary engineering challenge:

- **Software engineering** — deployment, APIs, infrastructure
- **Data engineering** — pipelines, storage, versioning
- **Distributed systems** — scalability, availability, consistency, fault tolerance
- **Product design** — stakeholder requirements, user experience
- **Operations** — monitoring, maintenance, incident response
- **Model lifecycle management** — experimentation, retraining, evolution
- **Governance** — compliance, privacy, regulatory requirements

Today: Map the problem space, explore key architectural principles, and understand fundamental design trade-offs and constraints

Lecture 4: ML System Design and Architecture
"The [ML] algorithm is only a small part of an ML system in production."
— Chip Huyen, Designing Machine Learning Systems
Production ML is a multidisciplinary engineering challenge:
Software engineering — deployment, APIs, infrastructure
Data engineering — pipelines, storage, versioning
Distributed systems — scalability, availability, consistency, fault tolerance
Product design — stakeholder requirements, user experience
Operations — monitoring, maintenance, incident response
Model lifecycle management — experimentation, retraining, evolution
Governance — compliance, privacy, regulatory requirements
Today: Map the problem space, explore key architectural principles, and understand fundamental design trade-offs and constraints

MLOps (Machine Learning Operations)

"An ML engineering culture and practice that aims at unifying ML system development (Dev) and ML system operation (Ops)."

[Google Cloud, 2020]

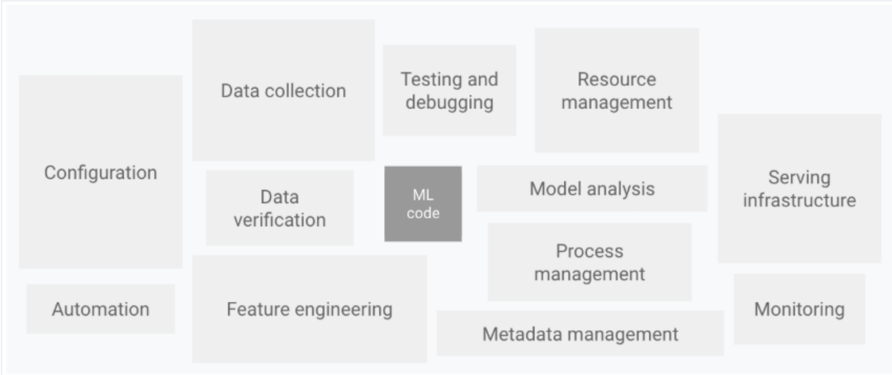
ML Systems Design

"Takes a system approach to MLOps [...] considers an ML system holistically to ensure that all the components and their stakeholders can work together to satisfy the specified objectives and requirements."

[Huyen, 2022]

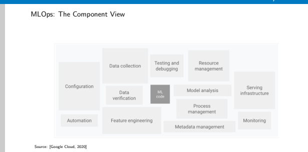
MLOps and ML Systems Design
MLOps (Machine Learning Operations)
"An ML engineering culture and practice that aims at unifying ML system development (Dev) and ML system operation (Ops)."
[Google Cloud, 2020]
ML Systems Design
"Takes a system approach to MLOps [...] considers an ML system holistically to ensure that all the components and their stakeholders can work together to satisfy the specified objectives and requirements."
[Huyen, 2022]

MLOps: The Component View



Source: [Google Cloud, 2020]

Lecture 4: ML System Design and Architecture
The Industrial ML Landscape & Course Roadmap
MLOps: The Component View
2025-10-16



Research ML vs Production ML

	Research	Production
Requirements	State-of-the-art model performance on benchmarks	Different stakeholders have different requirements
Computational priority	Fast training, high throughput	Fast inference, low latency
Data	Static	Constantly shifting
Fairness	Often not a focus	Must be considered
Interpretability	Often not a focus	Must be considered
Lifecycle	Projects have defined endpoints	Requires continuous operation

Adapted and extended from [Huyen, 2022], Table 1-1

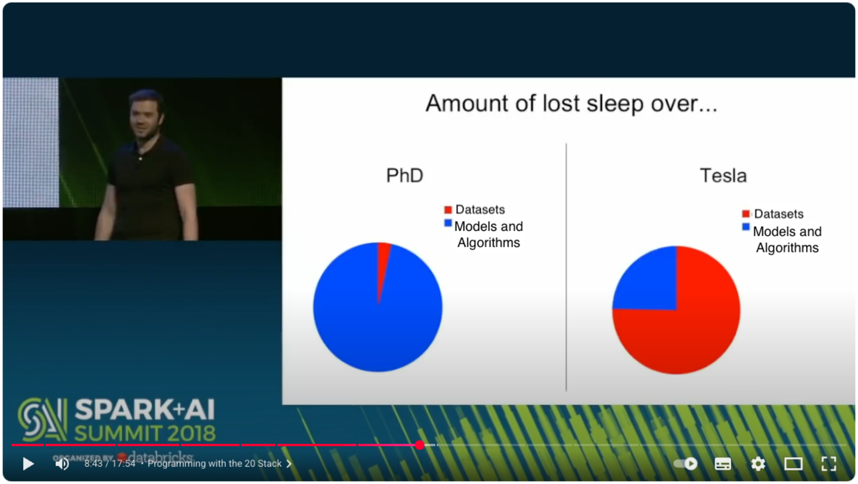
Note: Generalized comparison — trends vary by domain and exceptions exist.

Research ML vs Production ML

	Research	Production
Requirements	State-of-the-art model performance on benchmarks	Different stakeholders have different requirements
Computational priority	Fast training, high throughput	Fast inference, low latency
Data	Static	Constantly shifting
Fairness	Often not a focus	Must be considered
Interpretability	Often not a focus	Must be considered
Lifecycle	Projects have defined endpoints	Requires continuous operation

Adapted and extended from [Huyen, 2022], Table 1-1

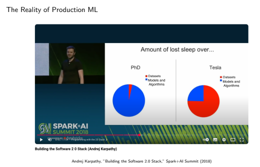
Note: Generalized comparison — trends vary by domain and exceptions exist.



Building the Software 2 0 Stack (Andrej Karpathy)

Andrej Karpathy, "Building the Software 2.0 Stack," Spark+AI Summit (2018)

Lecture 4: ML System Design and Architecture
└─The Industrial ML Landscape & Course Roadmap
└─The Reality of Production ML
2025-10-16



7 Years Later: Same Message With LLMs

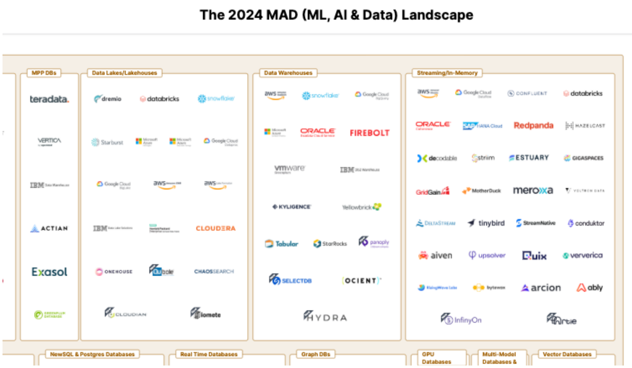
Context Enginnering over Prompt Engineering

"After a few years of prompt engineering being the focus of attention in applied AI, [...] Building with language models is becoming less about finding the right words and phrases for your prompts, and more about answering the broader question of "what configuration of context is most likely to generate our model's desired behavior?""

Source: Anthropic Engineering Blog (September 2025)

7 Years Later: Same Message With LLMs
Context Enginnering over Prompt Engineering
"After a few years of prompt engineering being the focus of attention in applied AI, [...] Building with language models is becoming less about finding the right words and phrases for your prompts, and more about answering the broader question of "what configuration of context is most likely to generate our model's desired behavior?""
Source: Anthropic Engineering Blog (September 2025)

The ML/AI Ecosystem: 2024 Landscape



- **2,011 companies** in the 2024 MAD (Machine Learning, AI & Data) Landscape
- Overwhelming number of alternatives
- We focus on core concepts rather than specific tool recommendations

Source: https://mad.firstmark.com/ (FirstMark)

The ML/AI Ecosystem: 2024 Landscape
The 2024 MAD (ML, AI & Data) Landscape
2,011 companies in the 2024 MAD (Machine Learning, AI & Data) Landscape
Overwhelming number of alternatives
We focus on core concepts rather than specific tool recommendations
Source: https://mad.firstmark.com/ (FirstMark)

Course Structure: From Model Development to Production

Today's transition

We begin a lecture block where we transition from *how to build models* to *how to build systems that run models reliably at scale*

Lecture Overview

Week	Date	Topic
1	Sep, 23 (TB)	Introduction
2	Sep, 30 (TB)	Classical MLE tools and methods, datasets
3	Oct, 07 (TB)	Deep MLE tools and methods, datasets
4	Oct, 14 (JB)	**ML System Design and Architecture**
5	Oct, 21 (JL)	**Data storage frameworks**
-	Oct, 28 (-)	Canceled - National holiday
6	Nov, 04 (JL)	**Machine learning model execution paradigms**
7	Nov, 11 (JB+TB)	**Ground truth management**
8	Nov, 18 (JB)	**Production metrics and observability**
9	Nov, 25 (JB)	**ML and AI technical debt**
10	Dec, 02 (TB)	AI engineering, MCP
11	Dec, 09 (TB)	Containerization (Docker, Apptainer)
12	Dec, 16 (TB)	Development workflows (git, CI-CD, BDD)
13	Jan, 06 (TB)	MLE and AI on the "edge"

Lecture 4: ML System Design and Architecture
The Industrial ML Landscape & Course Roadmap
Course Structure: From Model Development to Production
2025-10-16

Systems Must Survive Dynamic Environments (1)

Most software systems operate in changing environments

- User requirements evolve
- Business logic changes
- Load patterns shift
- Dependencies update
- External systems change

This drives core software engineering practices: Agile development, CI/CD, monitoring, incident response

Systems Must Survive Dynamic Environments (1)

Most software systems operate in changing environments

- User requirements evolve
- Business logic changes
- Load patterns shift
- Dependencies update
- External systems change

This drives core software engineering practices: Agile development, CI/CD, monitoring, incident response

Systems Must Survive Dynamic Environments (2)

ML systems face additional change vectors:

- **Data distribution shifts** — input patterns change over time (covariate shift)
- **Concept drift** — the relationship between inputs and outputs evolves
- **Feedback loops** — model predictions influence future training data, often implicitly unlike explicit control flows in traditional software
- **Adversarial dynamics** — in domains like fraud detection or security, adversaries adapt based on model decisions
- **Silent degradation** — performance declines gradually without explicit failures
- **Limited interpretability** — complex models function as black boxes, unlike traditional software where logic is directly inspectable in code
- **Development more experimental** - trying new techniques should be quick and we need to keep track of what worked and how well.

Systems Must Survive Dynamic Environments (2)

ML systems face additional change vectors:

- **Data distribution shifts** — input patterns change over time (covariate shift)
- **Concept drift** — the relationship between inputs and outputs evolves
- **Feedback loops** — model predictions influence future training data, often implicitly unlike explicit control flows in traditional software
- **Adversarial dynamics** — in domains like fraud detection or security, adversaries adapt based on model decisions
- **Silent degradation** — performance declines gradually without explicit failures
- **Limited interpretability** — complex models function as black boxes, unlike traditional software where logic is directly inspectable in code
- **Development more experimental** - trying new techniques should be quick and we need to keep track of what worked and how well.

Scope: What Systems Are We Discussing?

We speak in context of:

- Industry-scale systems with production workloads and real users
- Team-developed and operated systems requiring cross-functional collaboration
- Systems under load that justifies infrastructure investment and complexity

Important caveats:

- Design principles scale down, but not all patterns are needed at smaller scale
- Edge and embedded ML have different constraints (covered in Lecture 13)

Scope: What Systems Are We Discussing?

We speak in context of:
- Industry-scale systems with production workloads and real users
- Team-developed and operated systems requiring cross-functional collaboration
- Systems under load that justifies infrastructure investment and complexity

Important caveats:
- Design principles scale down, but not all patterns are needed at smaller scale
- Edge and embedded ML have different constraints (covered in Lecture 13)

Key Operational Constraints

Every design decision is constrained by requirements that are often in tension

Response Time Requirements

How fast must the system respond?

- **Synchronous (ms to $\pm$ 3s)**: Search, fraud detection, recommendations
  - Models pre-loaded, over-provision for peak load, often simpler models
- **Asynchronous (seconds to minutes)**: Spam filtering, content moderation
  - Workers scale with queue depth, more cost-efficient than synchronous
- **Batch (hours to days)**: Churn prediction, credit scoring
  - Load models on-demand, efficient batching, resources released after completion

Throughput Needs

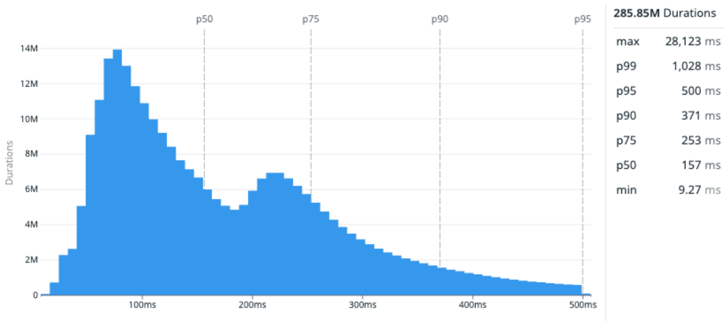
How much load must the system handle?

- Requests per second, predictions per day
- Peak vs. average load patterns (e.g., regular email volume spikes)

Key Operational Constraints
Every design decision is constrained by requirements that are often in tension
Response Time Requirements
How fast must the system respond?
Synchronous (ms to ± 3s): Search, fraud detection, recommendations
Models pre-loaded, over-provision for peak load, often simpler models
Asynchronous (seconds to minutes): Spam filtering, content moderation
Workers scale with queue depth, more cost-efficient than synchronous
Batch (hours to days): Churn prediction, credit scoring
Load models on-demand, efficient batching, resources released after completion
Throughput Needs
How much load must the system handle?
Requests per second, predictions per day
Peak vs. average load patterns (e.g., regular email volume spikes)

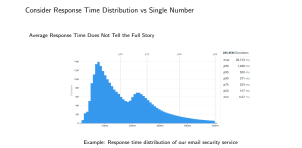
- Synchronous details:
  - Request blocks waiting for response - user/system actively waiting
  - Delay directly impacts user experience
  - Must keep instances warm (no cold starts tolerated)
  - Over-scale for traffic spikes (worse utilization: 30-40% average)
  - Higher infrastructure costs: paying for idle capacity, need redundancy
  - Optimization goal: responsiveness over efficiency
- Asynchronous details:
  - Processing triggered but not blocking
  - Request submitted to queue/stream; results consumed when ready
  - Can tolerate cold starts since nothing blocking
  - Scale workers based on queue depth (better utilization)
  - Allows more complex models without keeping them constantly warm
  - Middle ground between sync and batch
- Batch details:
  - Scheduled processing of accumulated data
  - Results computed offline and stored for later use
  - Can use large/complex models - load them on-demand
  - Process many predictions together (efficient batching, GPU utilization)
  - Shut down resources after completion (can achieve 100% utilization during run)
  - Cost-optimized for throughput over latency
  - Often uses MORE total compute but more efficiently packed
- Key economic insight: Latency-sensitive = pay for availability and spare capacity. Batch = pay for total compute but optimize utilization. It's not architecturally impossible to serve batch via sync API - it's economically wasteful.

Average Response Time Does Not Tell the Full Story



Example: Response time distribution of our email security service

Lecture 4: ML System Design and Architecture
ML System Design Fundamentals
Consider Response Time Distribution vs Single Number
2025-10-16



Two approaches to handle increased load:

Vertical Scaling

- Bigger machines (CPU, RAM, GPU)
- Reduces latency
- Hardware ceiling

Horizontal Scaling

- More machines in parallel
- Increases throughput
- No practical limit

Capacity in cloud is effectively unbounded — the constraint is cost, not capability

- Load peaks and low response time requirements drive up costs
- Faster startup times enable more efficient scaling strategies
- Design optimizes for unit economics: **$ per prediction**, **$ per user/year**

Vertical vs Horizontal Scaling
Two approaches to handle increased load:
Vertical Scaling
- Bigger machines (CPU, RAM, GPU)
- Reduces latency
- Hardware ceiling
Horizontal Scaling
- More machines in parallel
- Increases throughput
- No practical limit
Capacity in cloud is effectively unbounded — the constraint is cost, not capability
- Load peaks and low response time requirements drive up costs
- Faster startup times enable more efficient scaling strategies
- Design optimizes for unit economics: $ per prediction, $ per user/year

CPU vs. GPU for Inference

CPU inference:

- Simpler deployment (no drivers, standard tooling, works everywhere)
- Default for serverless (Lambda, SageMaker Serverless, Cloud Functions)
- Cost-effective for low or unpredictable traffic patterns
- Models are often made CPU-viable: INT8 quantization on Llama/Mistral improves throughput 25-45% [AMD, 2024]

GPU inference:

- Heavier models
- Batch processing amortizes costs
- High sustained throughput

Trade-off evolves as models and hardware improve, but remains.

AWS SageMaker Serverless Inference: CPU-only [Amazon Web Services, 2025]. Google Cloud Run added GPU support 2025 [Google Cloud, 2025].

CPU vs. GPU for Inference

CPU inference:

- Simpler deployment (no drivers, standard tooling, works everywhere)
- Default for serverless (Lambda, SageMaker Serverless, Cloud Functions)
- Cost-effective for low or unpredictable traffic patterns
- Models are often made CPU-viable: INT8 quantization on Llama/Mistral improves throughput 25-45% [AMD, 2024]

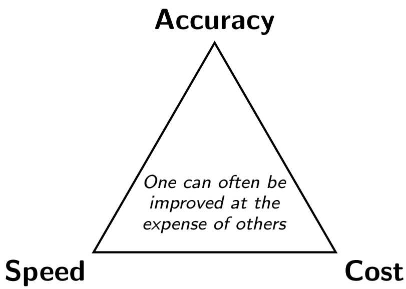
GPU inference:

- Heavier models
- Batch processing amortizes costs
- High sustained throughput

Trade-off evolves as models and hardware improve, but remains.

AWS SageMaker Serverless Inference: CPU-only [Amazon Web Services, 2025]. Google Cloud Run added GPU support 2025 [Google Cloud, 2025].

The Fundamental Trade-offs

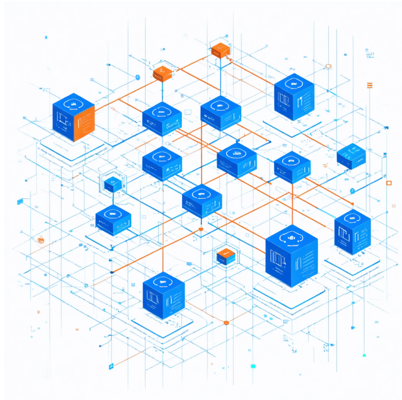


- **Search autocomplete**: Speed critical → accept accuracy ceiling, pay for infrastructure
- **Medical diagnosis**: Accuracy paramount → tolerate longer processing, higher compute costs
- **Email subject line suggestions**: Cost-effective + speed → accept "good enough" accuracy
- **Credit scoring**: Accuracy + cost optimized → batch processing relaxes speed constraint

The Fundamental Trade-offs
Accuracy
One can often be improved at the expense of others
Speed
Cost
- **Search autocomplete**: Speed critical → accept accuracy ceiling, pay for infrastructure
- **Medical diagnosis**: Accuracy paramount → tolerate longer processing, higher compute costs
- **Email subject line suggestions**: Cost-effective + speed → accept "good enough" accuracy
- **Credit scoring**: Accuracy + cost optimized → batch processing relaxes speed constraint

Modular ML System Design

Separate concerns to manage complexity and enable independent evolution



Key boundaries:

- Data pipelines — ingestion, validation, storage
- Feature engineering — computation, storage
- Training — experimentation, model selection
- Serving — inference, scaling, monitoring

Benefits:

- Independent evolution of components
- Isolated testing and debugging
- Team scaling through component ownership
- Reusability across models and use cases

Critical as systems grow — early systems can be monolithic

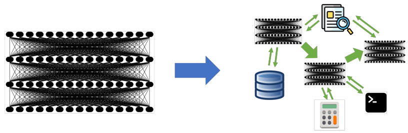
Modular ML System Design
Separate concerns to manage complexity and enable independent evolution
Key boundaries:
- Data pipelines — ingestion, validation, storage
- Feature engineering — computation, storage
- Training — experimentation, model selection
- Serving — inference, scaling, monitoring
Benefits:
- Independent evolution of components
- Isolated testing and debugging
- Team scaling through component ownership
- Reusability across models and use cases
Critical as systems grow — early systems can be monolithic

Compound AI Systems

Definition: Systems combining multiple interacting components to solve AI tasks – [Zaharia et al., 2024]

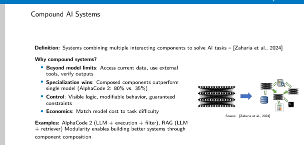
Why compound systems?

- **Beyond model limits**: Access current data, use external tools, verify outputs
- **Specialization wins**: Composed components outperform single model (AlphaCode 2: 80% vs. 35%)
- **Control**: Visible logic, modifiable behavior, guaranteed constraints
- **Economics**: Match model cost to task difficulty



Source: [Zaharia et al., 2024]

Examples: AlphaCode 2 (LLM + execution + filter), RAG (LLM + retriever) Modularity enables building better systems through component composition


Compound AI Systems
Definition: Systems combining multiple interacting components to solve AI tasks – [Zaharia et al., 2024]
Why compound systems?
Beyond model limits: Access current data, use external tools, verify outputs
Specialization wins: Composed components outperform single model (AlphaCode 2: 80% vs. 35%)
Control: Visible logic, modifiable behavior, guaranteed constraints
Economics: Match model cost to task difficulty
Examples: AlphaCode 2 (LLM + execution + filter), RAG (LLM + retriever) Modularity enables building better systems through component composition

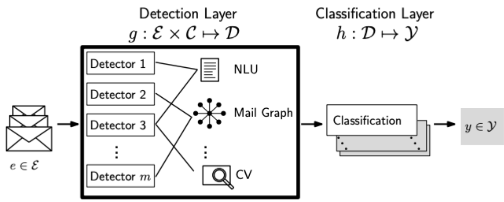
CAPE: Business Email Compromise detection system (production 5+ years, Cisco) – [Brabec et al., 2023]

Architecture:

- **Detection layer**: 90+ independent detectors
  - NLU models (urgency, call-to-action)
  - Computer vision (OCR, image analysis)
  - Graph-based communication patterns
- **Classification layer**: Model combining detections

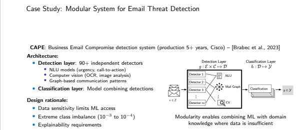
Design rationale:

- Data sensitivity limits ML access
- Extreme class imbalance (10^{-3} to 10^{-4})
- Explainability requirements

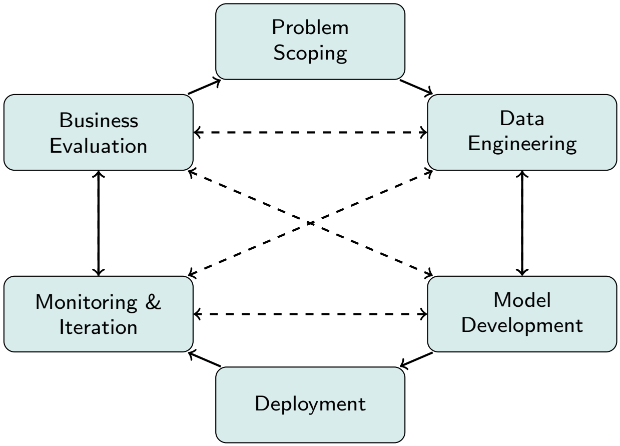


Modularity enables combining ML with domain knowledge where data is insufficient

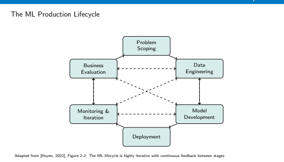
Lecture 4: ML System Design and Architecture
└─ML System Design Fundamentals
└─Case Study: Modular System for Email Threat Detection
2025-10-16



The ML Production Lifecycle



Adapted from [Huyen, 2022], Figure 2-2: The ML lifecycle is highly iterative with continuous feedback between stages



Maturity reflects velocity of deploying new models and level of automation [Google Cloud, 2020]

Level	Characteristics	Suitable When
Level 0 **Manual**	Notebooks, manual retraining, ad-hoc deployment, script-driven process	Models rarely change or retrain; sufficient for early ML adoption
Level 1 **CT Pipeline**	Automated ML pipeline, continuous training (CT), modularized components, pipeline deployment	Frequent retraining needed, data changes often, ML approach stable
Level 2 **CI/CD/CT**	Full automation: continuous integration, deployment, and training; automated testing and validation	Rapid experimentation, frequent pipeline changes, multiple models in production

- **Level 1**: "My data changes frequently, I need to retrain often" $\longrightarrow$ automate retraining
- **Level 2**: "My ML approach changes frequently, I need to rapidly deploy pipeline changes" $\longrightarrow$ automate pipeline updates

CT = Continuous Training (unique to ML); CI/CD details in Lecture 12

MLOps Maturity Levels

Maturity reflects velocity of deploying new models and level of automation [Google Cloud, 2020]

Level | Characteristics | Suitable When
Level 0 Manual | Notebooks, manual retraining, ad-hoc deployment, script-driven process | Models rarely change or retrain; sufficient for early ML adoption
Level 1 CT Pipeline | Automated ML pipeline, continuous training (CT), modularized components, pipeline deployment | Frequent retraining needed, data changes often, ML approach stable
Level 2 CI/CD/CT | Full automation: continuous integration, deployment, and training; automated testing and validation | Rapid experimentation, frequent pipeline changes, multiple models in production

Level 1: "My data changes frequently, I need to retrain often" → automate retraining
Level 2: "My ML approach changes frequently, I need to rapidly deploy pipeline changes" → automate pipeline updates
CT = Continuous Training (unique to ML); CI/CD details in Lecture 12

Deployment Strategies

How to safely release new models to production:

- **Shadow deployment**: New model runs in parallel, predictions logged but not used
  - Validate behavior before impact, compare against existing model
  - Cost: 2x inference compute during shadow period
- **Canary release**: Gradual rollout to increasing traffic percentage
  - Example: 5% → 25% → 50% → 100% over days/weeks
  - Monitor metrics at each stage, rollback if degradation detected
- **Blue-green deployment**: Instant switch between two production environments
  - Blue = current, Green = new; route traffic to Green, keep Blue ready
  - Fast rollback capability, but all-or-nothing risk
- **A/B testing**: Run both models simultaneously, compare business metrics
  - Randomized user assignment, measure impact on KPIs
  - Requires statistical rigor, longer duration for significance
  - Suitable for systems where we get feedback fast

Deployment Strategies

How to safely release new models to production:
- Shadow deployment: New model runs in parallel, predictions logged but not used
  - Validate behavior before impact, compare against existing model
  - Cost: 2x inference compute during shadow period
- Canary release: Gradual rollout to increasing traffic percentage
  - Example: 5% → 25% → 50% → 100% over days/weeks
  - Monitor metrics at each stage, rollback if degradation detected
- Blue-green deployment: Instant switch between two production environments
  - Blue = current, Green = new; route traffic to Green, keep Blue ready
  - Fast rollback capability, but all-or-nothing risk
- A/B testing: Run both models simultaneously, compare business metrics
  - Randomized user assignment, measure impact on KPIs
  - Requires statistical rigor, longer duration for significance
  - Suitable for systems where we get feedback fast

Team Organization: Horizontal vs. Vertical

Conway's Law: System architecture mirrors organizational communication structure

Two organizational axes:

- **Vertical (Embedded)**: ML practitioners within product/business teams
- **Horizontal (Centralized)**: ML practitioners in separate organization

Embedded Model

- Fast iteration
- Business context
- Clear ownership

Trade-off: Duplicated effort, inconsistent practices

Centralized Model

- Consistent tooling
- Knowledge sharing
- Specialization

Trade-off: Slower iteration, handovers, business disconnect

Most organizations at scale use **hybrid models**

2025-10-16
Lecture 4: ML System Design and Architecture
└─Production Lifecycle & Organization
└─Team Organization: Horizontal vs. Vertical

Team Organization: Horizontal vs. Vertical
Conway's Law: System architecture mirrors organizational communication structure
Two organizational axes:
Vertical (Embedded): ML practitioners within product/business teams
Horizontal (Centralized): ML practitioners in separate organization
Embedded Model
Fast iteration
Business context
Clear ownership
Trade-off: Duplicated effort, inconsistent practices
Centralized Model
Consistent tooling
Knowledge sharing
Specialization
Trade-off: Slower iteration, handovers, business disconnect
Most organizations at scale use hybrid models

Hybrid Model: Most Common at Scale

Combines horizontal infrastructure with vertical application teams

Typical structure:

1. **Platform/Infrastructure team** (centralized)
   - Builds shared ML tooling: pipelines, feature stores, model registry
   - Maintains standards and best practices
2. **Applied ML practitioners** (embedded in product teams)
   - Use platform tooling, own models end-to-end
   - Report functionally to ML org, operationally to product team
3. **Research/R&D team** (centralized) — *optional*
   - Forward-looking exploration, specialized domains (NLP, CV)

Examples: Google [Google Cloud, 2021], Cisco

Hybrid Model: Most Common at Scale
Combines horizontal infrastructure with vertical application teams
Typical structure:
1. Platform/Infrastructure team (centralized)
- Builds shared ML tooling: pipelines, feature stores, model registry
- Maintains standards and best practices
2. Applied ML practitioners (embedded in product teams)
- Use platform tooling, own models end-to-end
- Report functionally to ML org, operationally to product team
3. Research/R&D team (centralized) — optional
- Forward-looking exploration, specialized domains (NLP, CV)
Examples: Google [Google Cloud, 2021], Cisco

Research – Production Handoff Patterns

How do research/experimentation teams interact with production teams?

1. **"Throw over the wall" (Anti-pattern)**
   - Research team trains model, hands off to engineering for deployment
   - *Fails because*: Different incentives, knowledge loss, no ownership continuity
2. **End-to-end ownership**
   - Same team owns research through production
   - *Works when*: High ML maturity, strong engineering culture in DS team
   - *Challenge*: Requires DS with production skills or dual-skilled engineers
3. **Platform-based collaboration**
   - Shared tooling and processes rather than handoffs
   - Researchers use production-grade tools from day one
   - MLOps platform enables research → production without full handoff

Lecture 4: ML System Design and Architecture
└─Production Lifecycle & Organization
└─Research – Production Handoff Patterns
2025-10-16

Research – Production Handoff Patterns
How do research/experimentation teams interact with production teams?
"Throw over the wall" (Anti-pattern)
Research team trains model, hands off to engineering for deployment
Fails because: Different incentives, knowledge loss, no ownership continuity
End-to-end ownership
Same team owns research through production
Works when: High ML maturity, strong engineering culture in DS team
Challenge: Requires DS with production skills or dual-skilled engineers
Platform-based collaboration
Shared tooling and processes rather than handoffs
Researchers use production-grade tools from day one
MLOps platform enables research → production without full handoff

Key takeaways

- **Systems must survive dynamic environments** — ML adds change vectors beyond traditional software (data drift, feedback loops, silent degradation)
- **Design constraints** (response time, throughput, cost) are in tension — choose which to relax based on requirements
- **MLOps maturity evolves** as needs demand: Manual (L0) → Continuous Training (L1) → Full CI/CD (L2)
- **Team structure shapes architecture** — horizontal (centralized) vs. vertical (embedded); hybrid models combine both

Next: L5-9 cover storage, execution, ground truth, monitoring, technical debt; L12 covers CI/CD mechanics

Key takeaways

- **Systems must survive dynamic environments** — ML adds change vectors beyond traditional software (data drift, feedback loops, silent degradation)
- **Design constraints** (response time, throughput, cost) are in tension — choose which to relax based on requirements
- **MLOps maturity evolves** as needs demand: Manual (L0) → Continuous Training (L1) → Full CI/CD (L2)
- **Team structure shapes architecture** — horizontal (centralized) vs. vertical (embedded); hybrid models combine both

Next: L5-9 cover storage, execution, ground truth, monitoring, technical debt; L12 covers CI/CD mechanics

Thanks for your attention

Thanks for your attention

References I

Google Cloud. (2020). *Mlops: Continuous delivery and automation pipelines in machine learning* (tech. rep.). Google Cloud Architecture Center. https://cloud.google.com/architecture/mlops-continuous-delivery-and-automation-pipelines-in-machine-learning

Huyen, C. (2022). *Designing machine learning systems.* " O'Reilly Media, Inc.".

AMD. (2024, October). Leaner LLM inference with INT8 quantization on AMD GPUs using PyTorch [ROCm Blogs]. https://rocm.blogs.amd.com/artificial-intelligence/int8-quantization/README.html

Amazon Web Services. (2025). Deploy models with Amazon SageMaker serverless inference [Amazon SageMaker AI Documentation]. https://docs.aws.amazon.com/sagemaker/latest/dg/serverless-endpoints.html

Google Cloud. (2025).Google cloud run now offers serverless GPUs for AI and batch processing. *InfoQ.* https://www.infoq.com/news/2025/06/google-cloud-run-nvidia-gpu/

Zaharia, M., Khattab, O., Chen, L., Davis, J. Q., Miller, H., Potts, C., Zou, J., Carbin, M., Frankle, J., Rao, N., & Ghodsi, A. (2024). The shift from models to compound ai systems.

Brabec, J., Šrajer, F., Starosta, R., Sixta, T., Dupont, M., Lenoch, M., Menšík, J., Becker, F., Boros, J., Pop, T., et al. (2023).A modular and adaptive system for business email compromise detection. *arXiv preprint arXiv:2308.10776.*

Google Cloud. (2021). *Ai adoption framework* (tech. rep.). Google Cloud. https://services.google.com/fh/files/misc/ai_adoption_framework_whitepaper.pdf

Lecture 4: ML System Design and Architecture
References
References
2025-10-16
References I