ML SYSTEM DESIGN SERIES

NOTEBOOK থেকে PRODUCTION

MODEL

থেকে

SYSTEM

FROM A MODEL TO A PRODUCTION SYSTEM

01 MODEL → 02 FEATURES → 03 SERVING → 04 MONITORING

DATA • SCALABILITY • LATENCY • RELIABILITY • MLOPS

WRITTEN BY

Md. Aminul Haque Palash

Senior Software Engineer — AI/ML

BANGLA EDITION

Model থেকে System — Notebook থেকে Production

সূচীপত্র

1. ভূমিকা 4
    - 1.1. Model থেকে System 4
    - 1.2. একটি Recommendation System দিয়ে বুঝি 4
    - 1.3. কেন Interview-এ বিষয়টি গুরুত্বপূর্ণ? 5
    - 1.4. Traditional System Design-এর সঙ্গে পার্থক্য 6
    - 1.5. Technology নয়, Problem দিয়ে শুরু 6
    - 1.6. Production Engineering হলো Trade-off-এর খেলা 6
    - 1.7. Failure স্বাভাবিক ধরে Design করুন 7
    - 1.8. এই বই কার জন্য? 7
    - 1.9. কীভাবে বইটি পড়বেন? 7
    - 1.10. Chapter Summary 8
2. Scalability, Latency & Reliability Fundamentals 9
    - 2.1. Scalability — Workload বাড়লে System কীভাবে চলবে? 9
        - 2.1.1. ML System-এ Scalability 9
    - 2.2. Vertical vs Horizontal Scaling 9
        - 2.2.1. Vertical Scaling — Scale Up 9
        - 2.2.2. Horizontal Scaling — Scale Out 10
    - 2.3. Latency — একটি Response পেতে কত সময় লাগে? 10
        - 2.3.1. Average নয়, Percentile Latency 10
    - 2.4. Throughput — নির্দিষ্ট সময়ে কত কাজ হয়? 11
        - 2.4.1. Latency vs Throughput 11

1. ভূমিকা

Machine Learning শেখার শুরুতে আমরা সাধারণত dataset নিই, model train করি, accuracy দেখি এবং hyperparameter tune করি। ভালো score এলে মনে হয়—কাজ শেষ।

কিন্তু industry-তে দ্রুত বোঝা যায়, একটি ভালো **Machine Learning model** এবং একটি ভালো **Machine Learning system** এক জিনিস নয়।

Notebook-এ model 95% accuracy দিতে পারে, কিন্তু production-এ prediction দিতে দুই second সময় নিতে পারে। Training-এর feature real-time prediction-এর সময় না-ও পাওয়া যেতে পারে। Traffic বাড়লে database slow হতে পারে। Model service down হলে application-এর গুরুত্বপূর্ণ feature বন্ধ হতে পারে। আবার model এক মাস পরও আগের মতো কাজ করছে কি না—সেটিও track করতে হয়।

এই জায়গা থেকেই শুরু হয় **ML System Design & Production Engineering**।

> Production-এ প্রশ্ন শুধু "কোন model ব্যবহার করব?" নয়। প্রশ্ন হলো—model-টিকে কীভাবে fast, scalable, reliable এবং useful service হিসেবে চালানো হবে?

1.1. Model থেকে System

একটি production ML system design করতে সাধারণত এসব প্রশ্নের উত্তর দরকার:

- Prediction কত দ্রুত দিতে হবে?
- Peak traffic কত এবং traffic বাড়লে কী হবে?
- Online feature কোথা থেকে আসবে এবং কত fresh হতে হবে?
- Training ও serving-এ feature একইভাবে calculate হচ্ছে কি না?
- Model unavailable হলে fallback কী হবে?
- New model safely deploy এবং compare করা হবে কীভাবে?
- Data drift বা model performance degradation কীভাবে detect হবে?

এই প্রশ্নগুলোই notebook-level ML থেকে production engineering-কে আলাদা করে।

1.2. একটি Recommendation System দিয়ে বুঝি

ধরুন, একটি food delivery app-এর জন্য personalized restaurant recommendation system তৈরি করতে হবে। Model হিসেবে collaborative filtering, embedding বা ranking model ব্যবহার করা যেতে পারে। Offline evaluation-এ Recall@K বা NDCG মাপা যেতে পারে।

কিন্তু production system-এর আরও কিছু constraint আছে:

- Recommendation 150 ms-এর মধ্যে দেখাতে হবে।
- Platform-এ কয়েক মিলিয়ন user ও কয়েক লাখ restaurant/menu item আছে।
- Peak hour-এ প্রতি second-এ হাজার হাজার request আসে।
- শুধু open এবং user-এর location-এ deliverable restaurant দেখাতে হবে।
- Model fail করলেও home page blank রাখা যাবে না।

তাই প্রতিটি request-এ সব item score করা practical নয়। একটি সাধারণ online flow:

```
User Request
     ↓
```

```
API / Gateway
      ↓
User & Context Features
      ↓
Candidate Generation
      ↓
Availability Filters
      ↓
Ranking Model
      ↓
Business Rules
      ↓
Top Results
```

Background-এ data ও model lifecycle-ও চলে:

```
Clicks / Orders / Events
          ↓
     Data Pipeline
          ↓
   Feature Generation
          ↓
     Model Training
          ↓
   Offline Evaluation
          ↓
     Model Registry
          ↓
      Deployment
          ↓
      Monitoring
```

> Model পুরো system নয়; এটি data, feature, API, storage, infrastructure, business rules এবং monitoring-এর সঙ্গে যুক্ত একটি component।

1.3. কেন Interview-এ বিষয়টি গুরুত্বপূর্ণ?

ML Engineer, AI Engineer, Applied Scientist বা Senior Data Scientist role-এর interview-এ শুধু model train করার দক্ষতা দেখা হয় না। Interviewer জানতে চান candidate নিচের বিষয়গুলো বুঝতে পারেন কি না:

- scale এবং latency requirement;
- offline ও online computation-এর boundary;
- cache, queue এবং database-এর প্রয়োজন;
- safe deployment ও rollback strategy;
- failure handling ও graceful degradation;
- logging, monitoring এবং retraining loop;
- accuracy, latency, freshness, reliability ও cost-এর trade-off।

এই বইয়ের লক্ষ্য architecture মুখস্থ করানো নয়; problem দেখলে structured way-তে চিন্তা করার অভ্যাস তৈরি করা।

1.4. Traditional System Design-এর সঙ্গে পার্থক্য

Traditional System Design-এর database, cache, load balancer, message queue, replication, sharding এবং distributed systems—সবই ML Engineer-এর জন্য দরকারি। ML system-এ এর সঙ্গে আরও কিছু concern যোগ হয়:

- Feature computation ও freshness
- Offline এবং online feature consistency
- Training-serving skew
- Batch ও online inference
- Candidate generation এবং ranking
- Model registry ও deployment
- Data drift এবং concept drift
- Feedback loop, delayed label ও retraining

তাই একই recommendation architecture-এ Backend Engineer API ও database scaling নিয়ে ভাববেন; ML Engineer-কে একই সঙ্গে candidate, feature freshness, model quality এবং feedback bias নিয়েও ভাবতে হবে।

1.5. Technology নয়, Problem দিয়ে শুরু

Requirement না বুঝে “Kafka, Redis, Kubernetes, Spark” বলা একটি common mistake। Technology-এর নাম architecture-এর justification নয়। প্রথমে জানতে হবে:

- Traffic ও data volume কত?
- Real-time result সত্যিই দরকার কি না?
- Acceptable latency ও availability কত?
- কোন failure business-এর জন্য সবচেয়ে ক্ষতিকর?

```
Requirement
     ↓
Constraint
     ↓
Trade-off
     ↓
Architecture
     ↓
Technology
```

Problem ও constraint পরিষ্কার হলে উপযুক্ত technology বেছে নেওয়া সহজ হয়।

1.6. Production Engineering হলো Trade-off-এর খেলা

ধরুন, দুটি model আছে:

```
Model A                          Model B
Accuracy = 94%                   Accuracy = 95%
Latency  = 40 ms                 Latency  = 900 ms
Cost     = Low                   Cost     = 8× higher
```

কোনটি ভালো—এর উত্তর সবসময় “Model B” নয়। Fraud detection-এ 1% improvement-এর financial value বড় হতে পারে। কিন্তু search autocomplete-এ 900 ms latency user experience নষ্ট করতে পারে। সিদ্ধান্ত business goal ও constraint-এর উপর নির্ভর করে।

Production ML-এ common trade-off:

- Accuracy বনাম Latency
- Latency বনাম Throughput
- Freshness বনাম Cost
- Recall বনাম Serving Time
- Personalization বনাম Cacheability
- Reliability বনাম Infrastructure Cost
- Complexity বনাম Maintainability

Strong interview answer শুধু final architecture বলে না; কোন trade-off-এর কারণে সেটি বেছে নেওয়া হয়েছে, তাও ব্যাখ্যা করে।

1.7. Failure স্বাভাবিক ধরে Design করুন

Production-এ server crash, network timeout, database lag, missing feature বা external API failure ঘটতে পারে। তাই একটি গুরুত্বপূর্ণ প্রশ্ন:

> এই component fail করলে user কী দেখবে এবং system কীভাবে চলবে?

উদাহরণ:

- Recommendation model unavailable → user-এর area-র popular items
- ETA model unavailable → map ETA + historical median
- Fraud model unavailable → rule-based risk check

একে **Graceful Degradation** বলা হয়। Full intelligence পাওয়া না গেলেও service যেন একটি সীমিত কিন্তু useful অবস্থায় থাকে।

1.8. এই বই কার জন্য?

এই বই Machine Learning Engineer, AI Engineer, Data Scientist, Applied ML Engineer, ML-focused Software Engineer এবং production ML শিখতে আগ্রহী student-দের জন্য। Basic Machine Learning জানা থাকলে discussion follow করা সহজ হবে; distributed systems-এর advanced knowledge শুরুতেই প্রয়োজন নেই।

1.9. কীভাবে বইটি পড়বেন?

Definition মুখস্থ না করে প্রতিটি concept-কে বাস্তব system-এর সঙ্গে মিলিয়ে দেখুন:

- Traffic 10× হলে প্রথম bottleneck কোথায় হবে?
- Model 500 ms নিলে 100 ms target কীভাবে পূরণ করবেন?
- কোন result cache করা যাবে?
- Training ও serving feature আলাদা হলে কী সমস্যা হবে?
- Input data বদলেছে, নাকি model খারাপ হয়েছে—কীভাবে বুঝবেন?

Food Delivery, Search, Recommendation, Fraud, ETA বা Chatbot—যেকোনো পরিচিত system ধরে চিন্তা করলে conceptগুলো connected architecture হিসেবে ধরা দেবে।

Interview question পেলে এই flow ব্যবহার করুন:

```
1. Requirement ও scope
2. Scale ও constraints
3. ML objective ও metrics
4. Data ও feature pipeline
5. Model training ও evaluation
6. Online serving ও latency
7. Scaling ও storage
8. Failure ও fallback
9. Logging ও monitoring
10. Deployment ও retraining
```

এটি exact answer নয়; discussion organized রাখার framework।

1.10. Chapter Summary

Machine Learning শেখায় কীভাবে prediction তৈরি করতে হয়। ML System Design শেখায় সেই prediction-কে বাস্তব পৃথিবীতে কীভাবে fast, scalable, reliable এবং maintainable service-এ পরিণত করতে হয়।

পরবর্তী chapter-এ আমরা শুরু করব চারটি fundamental concern দিয়ে: **Scalability, Latency, Throughput এবং Reliability**।

2. Scalability, Latency & Reliability Fundamentals

একটি production system design করার আগে চারটি প্রশ্ন পরিষ্কার করতে হয়:

- Workload বাড়লে system কীভাবে scale করবে?
- একটি request-এর response কত দ্রুত আসবে?
- নির্দিষ্ট সময়ে system কত request handle করতে পারবে?
- Failure হলেও service কতটা dependable থাকবে?

এই chapter-এ Availability, Fault Tolerance, High Availability এবং SLI–SLO–SLA-এর সম্পর্কও সহজভাবে দেখা হবে।

2.1. Scalability — Workload বাড়লে System কীভাবে চলবে?

Business বড় হলে user, request ও data-এর পরিমাণ বাড়ে। এই workload acceptable latency, reliability ও cost-এর মধ্যে handle করার ক্ষমতাই **Scalability**।

```
1,000 requests/minute
          ↓ Business Growth
100,000 requests/minute
```

শুধু আরও server যোগ করলেই system scalable হয় না। Algorithm, database, data pipeline এবং ML inference path-কেও বড় workload-এর উপযোগী হতে হয়।

2.1.1. ML System-এ Scalability

5 million products থাকলে প্রতিটি request-এ সব product heavy model দিয়ে score করা slow ও expensive। তাই কাজটি সাধারণত দুই stage-এ ভাগ করা হয়:

```
5 Million Products
        ↓
Candidate Generation
        ↓
500 Relevant Candidates
        ↓
Ranking Model
        ↓
Top 20 Recommendations
```

Candidate generation দ্রুত relevant set বের করে। Expensive ranking model শুধু সেই set score করে। এখানে scalability infrastructure এবং algorithm—দুই জায়গায়।

Interview-ready Definition

> Scalability হলো users, data বা traffic বাড়লেও acceptable latency, reliability এবং cost বজায় রেখে workload handle করার ক্ষমতা।

2.2. Vertical vs Horizontal Scaling

Capacity বাড়ানোর দুটি common উপায় আছে।

2.2.1. Vertical Scaling — Scale Up

Existing machine-এর CPU, RAM বা hardware capacity বাড়ানোকে Vertical Scaling বলে।

```
8 CPU, 16 GB RAM
        ↓ Upgrade
64 CPU, 256 GB RAM
```

এটি তুলনামূলক সহজ। কিন্তু একটি machine-এর physical limit আছে, high-end hardware-এর cost দ্রুত বাড়ে এবং machine fail করলে single point of failure হতে পারে।

2.2.2. Horizontal Scaling — Scale Out

একটি machine শক্তিশালী করার বদলে আরও machine যোগ করা Horizontal Scaling।

```
Users
  ↓
Load Balancer
  ↓
S1    S2    S3
```

Traffic বাড়লে server যোগ করা যায়। তবে request distribution, shared state, data consistency, failure handling ও database bottleneck-এর মতো distributed-system complexity তৈরি হয়।

> **Vertical Scaling** → Machine শক্তিশালী করা।
> **Horizontal Scaling** → Machine-এর সংখ্যা বাড়ানো।

Large-scale system-এ horizontal scaling বেশি flexible, কিন্তু complexity বেশি। অনেক architecture বাস্তবে দুই পদ্ধতিই ব্যবহার করে।

2.3. Latency — একটি Response পেতে কত সময় লাগে?

Request পাঠানো থেকে response ফিরে আসা পর্যন্ত সময়কে **Latency** বলা হয়।

```
Request
   ↓
Recommendation API
   ↓
80 ms
   ↓
Response
```

Model accurate হলেও required time-এর মধ্যে prediction দিতে না পারলে online use case-এ সেটি practical নাও হতে পারে।

```
Model A                          Model B
Quality = Very High               Quality = Slightly Lower
Latency = 5 seconds               Latency = 100 ms
```

Interactive recommendation-এর জন্য Model B বেশি useful হতে পারে। তাই quality এবং response speed একসঙ্গে evaluate করতে হয়।

2.3.1. Average নয়, Percentile Latency

Average latency slow user experience লুকাতে পারে। তাই p50, p95 ও p99 দেখা হয়।

```
p50 = 80 ms   → 50% request 80 ms বা কম
p95 = 300 ms  → 95% request 300 ms বা কম
p99 = 900 ms  → 99% request 900 ms বা কম
```

p95 ও p99 slow requests বা **tail latency** বুঝতে সাহায্য করে। Percentile-এর সঙ্গে time window এবং traffic volume-ও উল্লেখ করা উচিত।

Interview-ready Definition

> Latency হলো একটি request process করে response ফিরিয়ে দিতে system-এর যে সময় লাগে। Production-এ average-এর পাশাপাশি p50, p95 ও p99 দেখা হয়।

2.4. Throughput — নির্দিষ্ট সময়ে কত কাজ হয়?

Throughput হলো একটি system নির্দিষ্ট সময়ে মোট কত request বা task process করতে পারে। Web service-এ এটি সাধারণত RPS—Requests Per Second—দিয়ে প্রকাশ করা হয়।

```
Throughput = 10,000 requests/second
           = 10,000 RPS
```

2.4.1. Latency vs Throughput

> **Latency** ⟶ একটি request কত দ্রুত complete হয়?
> **Throughput** ⟶ নির্দিষ্ট সময়ে মোট কত request complete হয়?

Restaurant-এ একজন customer খাবার পেতে 10 মিনিট অপেক্ষা করলে সেটি latency-এর মতো। Restaurant এক ঘণ্টায় 300 order serve করলে সেটি throughput-এর মতো।

Capacity-এর বেশি traffic এলে queue তৈরি হয়; তখন throughput limit-এর কারণে observed latency-ও বাড়ে। ML serving-এ batching throughput বাড়াতে পারে, কিন্তু batch পূর্ণ হওয়ার অপেক্ষা latency বাড়াতে পারে।

Interview-ready Definition

> Throughput হলো একটি system নির্দিষ্ট সময়ের মধ্যে মোট কত request বা task process করতে পারে। Web system-এ এটি সাধারণত RPS দিয়ে প্রকাশ করা হয়।

ABOUT THE AUTHOR

লেখক পরিচিতি

Md. Aminul Haque Palash

Software Engineer • Machine Learning Practitioner

Md. Aminul Haque Palash একজন Software Engineer এবং Machine Learning practitioner। তিনি Chittagong University of Engineering and Technology (CUET) থেকে Computer Science and Engineering (CSE) বিষয়ে B.Sc. ডিগ্রি অর্জন করেছেন এবং software engineering ও machine learning-এর সমন্বয়ে বাস্তবধর্মী AI solution তৈরিতে কাজ করে আসছেন।

তাঁর কাজের অন্যতম প্রধান আগ্রহ হলো Machine Learning model-কে বাস্তব production environment-এ কার্যকরভাবে ব্যবহারের উপযোগী করে তোলা। তাঁর দৃষ্টিতে একটি ML system-এর সাফল্য শুধু model accuracy দিয়ে নির্ধারিত হয় না; এর সঙ্গে scalability, latency, data quality, reliability, monitoring এবং বাস্তব business requirements-ও সমান গুরুত্বপূর্ণ।

বাংলাভাষী engineers ও শিক্ষার্থীদের জন্য **ML System Design** এবং **Production Engineering** নিয়ে সহজ, structured এবং interview-oriented learning resource তুলনামূলকভাবে সীমিত। এই প্রয়োজন থেকেই বইটি লেখার উদ্যোগ। বইটির লক্ষ্য শুধু বিভিন্ন technical term বা framework মুখস্থ করানো নয়; বরং প্রতিটি concept কেন প্রয়োজন, এর পেছনের engineering trade-off কী এবং একটি real-world production system-এ এগুলো কীভাবে একসঙ্গে কাজ করে—তা সহজ ও practical উপায়ে তুলে ধরা।

এই বইয়ের মাধ্যমে পাঠক Machine Learning system-কে শুধু একটি model হিসেবে নয়, বরং **data থেকে deployment, monitoring এবং continuous improvement পর্যন্ত একটি সম্পূর্ণ engineering system** হিসেবে দেখার সুযোগ পাবেন।