ML System Design Interview

(Alex Xu et al.)

1. Clarifying requirements (questions intentionally vague)

a) **business objective** (recommender system for vacation rentals – increase # bookings / revenue),
b) system's **features to support** (video recommendations – like / dislike button to label data),
c) **data** – sources, size, labeled
d) **constraints** – available compute power, in cloud or on edge device, should model auto-improve over time?
e) **scale** of system - # users, # videos, their rate of growth
f) **performance** – prediction speed, real-time solution, accu or latency more important?

2. Framing ML Task

E.g. interviewer – increase user engagement, but it's not ML task.
a) **ML necessary**?
b) **translate business objective into ML objective** (improve social media safety => accurately predict harmful content, increase user engagement => maximize time users watch videos) / pros and cons of different objectives
c) **inputs and outputs** (for each model), different diagrams/ways to specify them are possibles
d) **ML algo type**: [un]supervised, binary / multiclass / multilabel classif. (discrete labels)/regression (w/output range)(predict continuous values), clustering/dimensionality reduction, association, RL.

3. Data Preparation

a) Data Eng.

- **data sources** – what data available, user generated or system generated, who collected, can we trust data? how clean is data? how often it comes in
- Is **privacy** / PII a concern? *Anonymization* needed? *Can we store data* or only in user's system?
- **data storage** – *where* is data stored? In could or devices? *Format*? How is multimodal data stored (text/image)? *RDBMS*, *key-value store* (Redis in-memory, DynamoDB), *columnar DB* (HBase, Cassandra – wide column store = 2D key-value store = names and format of cols can vary from row to row), *GraphDB* (Neo4J), *doc store* (MongoDB, CouchDB)
- **ETL** - **extract** from sources, **transform** – clean, map, re-format, **load** – to *DB*, *file*, *data warehouse* (refined struct. and semi-struct. data used for BI /reporting/analytics), *data lake* (raw unprocessed data – multimedia/logs/large files)
- **Data types** – *unstructured* (audio, video, image, text, no schema) & *structured* – numerical (discrete | continuous), categorical (ordinal | nominal)
- **Biases** in data? How to correct them?

b) Feature Eng.

- **Part I. Select predictive features** – domain knowledge, SMEs, automated methods (sklearn feature selection), feature importance, correlation matrix, PCA, LDA (lin discriminant an)
- **Part II. Everything below** - feature eng. operations
- **Missing values** – *remove* cols/rows (data loss), *imputation* by mean, median (adds noise)
- **Scaling** (different ranges in data) - *fit on trainset* to avoid data leakage!, normalize (min-max scaling {0-1]), standardize (0 mean, stdev 1), log scaling (mitigate skewness)
- **Bucketing** / discretization – putting continuous height or discrete age into categorical buckets

- **Encoding categorical features** – *integer encoding* as 1,2,3, etc. (if data has natural relationship), *one-hot encoding* if no relationship 100, 010, 001, etc., *embedding learning* or mapping to N-dimensional vectors if feature has many values (one-hot would have very large vectors here)
- **Build new features** – squares, cubes
- How **combine different data types**: text, nums, images – *feat union* or *different input layers* in DNN w/concatenation

4. Model development

a) Selection

- **Simple baseline** (model or workaround: most popular video as recommendation)
- Experiment w/**simple fast ML models** (LR, Decision Tree, RF, SVM, NB, XGBoost)
- **More complex models**: deep neural nets
- **Ensemble** of models: **Bagging** (bootstrap aggreg. = multiple models trained on *copies of data w/replacement*, then *averaging or voting*), **Boosting** (several simple *weak learners* - XGBost), **Stacking** (several weak learners + stacking model in the end uses their preds as feats), **Blending** (=stacking, but final models learns the val./test data in the end)
- When **choosing ML model** algo: review *trade-offs* (LR – linear task), *amount of data*, *training speed*, # HPs to tune, possibility of *continual learning* (online train, how often, personalize for each user), *compute requirements* (CPU, GPU), model *latency*, model *interpretability* (complex models – better performance, less interpretable), model for *edge device*? **model architecture** for DNNs – see below)

b) Training

- **Construct dataset**:
  * *collect* raw data, identify *features* (feat eng. step), identify *labels (manual labeling*, *natural* labeling (using likes/dislikes)), **split** data (avoid leakage), **sampling** strategy (*random*, *convenience* (who is convinient, nearby – cheap, but not generalizeable), snowball (research participants find other participants, rare samples (drug dealers), no generalizeable), *stratified random* (population divided into strate), *reservoir* (random sampling if u don't know the size of population), *importance* (sampling from alternative distribution / inference))
- **Imbalance** (resample, augment, loss f(x) to learn minority class better – class-balanced /focal loss – focus on misclassifieds)
- Choose **loss f(x)** – cross-entropy, hinge, (R)MSE, MAE, Huber
- Train **new model or fine-tune** (BERT, GPT)
- **Model architecture** for DNNs (# *layers*, # *neurons*, *activation f(x)* – Relu, Tanh, sigmoid, softmax, *optimization method* – Adam, RMSProp, SGD, AdaGrad, *momentum*, etc.)
- **Distributed training**: data parallelism, model parallelism
- **Overfitting / underfitting** concerns
- Do we get **user feedback**

5. Evaluation

a) Offline

- Train / val / test set

- **Classification** – multiclass/multilabel: prec, rec, F1, accu, confusion matrix; binary: ROC-AUC (TPR-FPR), PR-AUC (prec.-rec. curve)
- **Regression** – MSE, RMSE, MAE, Huber
- **Ranking** (recommend) – **prec@k**, **rec@k**, mean reciprocal rank **MRR** (1/position of first relevant item – *interpretable*!), mean ave. precision **mAP** (averages prec.@k at each relevant item position, rewards model that *puts more relevant stuff at top*, ideal=1, some interpretability), normalized discounted cumulative gain **nDCG** (cumulative gain = *sum of relevant items*, *discounted* for each position, *normalized* by position 1, *not interpretable* but takes into account *numeric relevancy*, tricky on practice => *not used often*)
- **Text generation** – **BLEU** (similarity of machine-transl. text to high quality translation = *precision-based n-gram overlap* w/brevity penalty), **ROUGE-N** (*recall-based n-gram overlap* in machine transl. or summary), **METEOR** (harmonic mean of *unigram precision and recall w/stemming and synonymy matching*, along w/exact word matching, fixes issues w/BLEU), **CIDEr** (Consensus-based Image Description Evaluation - evaluates for image Li *how well a candidate sentence Ci matches a set of image descriptions*), **SPICE** (Semantic Propositional *Image Caption Evaluation*)
- **Image generation** – **FID** (Frechet Inception Distance score - *distance between feature vectors* of real and generated images), **inception score** (takes a l*ist of GAN-generated images*, returns *one score 0-inf* re how good they are)

b) Online

- **Subjective** and depends on **business goals** (stakeholders)
- Ad click preds – **click-through rate (CTR) -** # people *who click* / # people *who see* ad, **revenue lift**
- Harmful content detect – valid appeal, prevalence
- Video recommend – CTR, **watch time**, **# watched** videos
- Friend recommend - **# requests** sent, # requests accepted

c) Other

- Is there **bias** across age, gender, race?
- What if users w/**malicious intents**?

6. Deployment and serving

a) If in could (*vs. on-device*):

- simpler deploy, more expensive, network latency,
- faster inference (compute-wise), fewer hardware constraints (memory, battery),
- less privacy, need Internet

b) GPU or CPU

- Cost of serving
- What other technologies improve speed and scalability of serving a model?

c) ML model consists of several components?

- Model *artifacts* (S3)
- Model *deploy instance* (GPU)
- *Orchestrator* instance (CPU)

- *Logging* service (splunk)
- *Data* to train / re-train (S3)

b) **Model compression**

- **Quantization** – use *fewer bits for model params* (reduces size) – usual single precision 32, half precision 16, fixed point 8 (int), 1-bit representation, *during* or *post training.* Downside: rounding error => performance change
- **Pruning** – setting *uncritical params to 0* => sparser model
- **Distillation** – small student model mimics teacher model

c) **Test in production**

- **Shadow deployment** – deploy *in parallel* w/existing model, but don't display preds to end user until new model is thoroughly tested; *costly* method – double predictions.
- **A/B testing** – also deploy in parallel, but part of traffic is routed to new model and the rest to old model (the bad part – lost data). Randomly select users and do statistical testing on a sufficient # data points
- **Canary release** - changes are initially released to a **small subset of users**
- I**nterleaving experiments** – **new & old results** are shown to same user who **ranks** them
- Multi-arm bandit test?

d) **Prediction pipeline**

- **Batch predictions** – preds are *pre-computed periodically* (speed unimportant). Drawbacks: often we *don't have data to predict in advance* (machine translation) + model *less sensitive to changing preferences* of users
- **Online predictions** – *immediate* preds (latency is a concern)
- Choice depends on requirements and trade-offs: preds needed in real time? Do we know what to pred in advance?

7. Monitoring and Infrastructure

Tracking, measuring, logging metrics to quickly fix ML system problems

- Common problem – **data distribution shift** (different than in training). To fix:
    - ✓ *Train on large datasets* (to cover as many data variations as possible)
    - ✓ *Augment* [image] data
    - ✓ Re-label new data and *retrain model*

What to monitor

- **Operational metrics**
    - ✓ Average serving time, Throughput
    - ✓ # pred requests, GPU/CPU utilization
- **ML metrics**
    - ✓ *Inputs / outputs,* Data *drift* / model drift
    - ✓ Model *accu* (target range), Model *versions*, *Log predictions*
- **Infrastructure**
    - ✓ *Location* (cloud or on-prem), *Compute* reqs (CPU or GPU)
    - ✓ *Network* infra (smooth operation), *Storage* infra (appropriate volume)
    - ✓ *Extend current infra* (if adding ML)
    - ✓ *Security*

Additional Topics (not from Alex Xu's book)

åData Distribution Shifts

- **Covariate shift** (input distrib. changes while conditional proba of output | input is same – demographics changes, but proba income same).
- **Label shift** (output distrib. changes, but input distrib. same) – share of pos labels doubled
- **Concept drift** (input data remains the same, but the output changes - changes in real estate market caused by the COVID-19 pandemic).

Detecting Data Distribution Shifts

- **Accuracy**, F1 score, recall, and AUC-ROC - requires ground truth (may not always be available)
- **Statistical methods** (mean, median, variance, two sample method - if differences between two sets of data are statistically significant).
- **Time scale windows** for detecting shifts (analyze **stats across various time windows** to identify when data changes occurred)

Addressing Data Distribution Shifts

1. Train models with **massive datasets** (cover all potential data points)
2. **Adapt pre-trained models** to a target distribution without requiring new labels
3. **Retrain model** with labeled data from the new target distribution – complicated: retrain from scratch or fine-tune? Which data to use for retraining?

Monitoring and Observability

- Monitoring - tracking, measuring, and logging metrics to identify errors.
  - Monitoring accuracy-related metrics
  - Monitoring preds, features, raw inputs
  - Setting up Logs, Dashboards, Alerts
- Observability - system setup for easier visibility and issue diagnosis

Continual Learning

- <u>Continual learning</u>, <u>monitoring</u>, and <u>testing in production</u> are vital to maintaining an adaptable ML system.
- Purpose - **adapt to evolving data distrib.**, online learning algos to update model params continuously w/new data
- Stateful training - **incremental learning** on new data.
- Stateless retraining - **training from scratch** each time.
- **Data iteration** - training from scratch or retraining the same model.
- **Model iteration** - adding <u>new features</u> or modifying <u>model's architecture</u>.

Infrastructure and Tooling for **MLOps**

- **Storage and Compute:** Amazon S3, CPU or GPU, AWS EC2, etc.
- Public Cloud vs. Company-Owned Data Centers
- Multicloud Strategy - minimize dependence on a single cloud provider

Dev Env

- IDE
- **Versioning** of code, parameters, and data
- CI/CD, GIT, Weights & Biases.
- MLflow for tracking

- Need to standardizing dev environments.
- **Docker containers** simplify deploy in prod – Dockerfile re-creates model run env => builds Docker image,
- Complex applications - multiple containers (featurizing on CPU and training on GPU)
- **Kubernetes** creates a scalable network for containers.

Resource Management

- **Cron programs** use DAGs to schedule and jobs based on event triggers
- **Schedulers** are concerned w/when and how to run jobs
- **Orchestrators** (Kubernetes) procure resources, handle lower-level abstractions such as machines, clusters, and replication, and can provision more computers.

- **Data Science Workflow Management**: Airflow, Argo, Prefect, Kubeflow, and Metaflow.
- **ML Platform**: Model Deployment, Model Store, Feature Store
- Build Versus Buy Decision

Add the most valuable insights from here:
https://www.trybackprop.com/blog/ml_system_design_interview

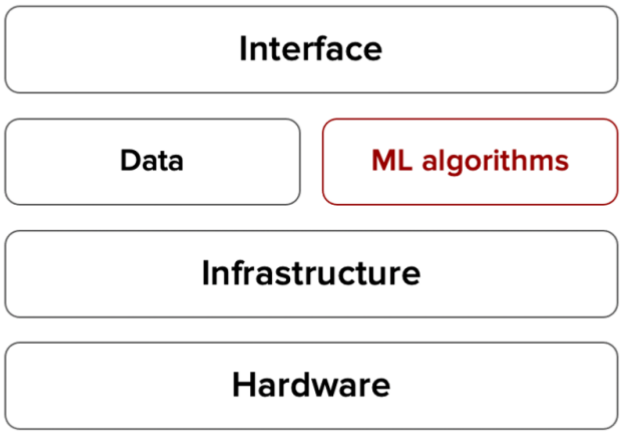
ML System Design – Other Sources (not processed)

Architecture and **infrastructure** for data collection, preprocessing, ML model training, evaluation, deployment, and inference.

ML System Qualities

- **Reliability** (sometimes there is no error message – only garbage output. E.g. meaningless translation).
- **Scalability –** autoscaling compute/storage (even 1 hr of downtime can cost $mlns – example: e-commerce website).
- **Maintainability -** identifying model drift/data drift, re-training.
- **Adaptability -** system should be flexible for quick updates without interruptions if new data is available or business objectives change.

Schematic Diagram for ML System



Steps in ML system design

1. ML problem definition
2. Collect relevant, representative data
3. Data preprocessing: clean, transform, normalize data
4. Models selection (ML algo, model architecture)
5. Training
6. Evaluation with appropriate metrics.
7. Deployment in production for real-world use
8. Monitoring model's performance, update when necessary, and maintain reliability and effectiveness

Data - data pipeline has appropriate **privacy controls** – e.g. PII should be properly handled - legal consequences.
Model training - model versioning for faster re-training;
Training must be reproducible (no major difference)
Serving models can be rolled back

Monitoring
feature generation code for both training and inference should be the same.
Handling invalid or implausible values correctly
Model / data drift
Computing performance has not regressed (serving latency, throughput).
Prediction quality has not regressed – re-train the model.
model monitoring: staleness - how often to re-train models if data distribution changes. Changes in ML system inputs should be captured to check for ML performance deterioration. If data drift - model re-trained.

Question to ask:

1. literally the first thing they do is start asking about numbers (throughput/latency requirements, data set sizes, "how much unlabelled data?", uptime, etc)
2. literally draw a stick figure to represent the user and keep going back to them to think about what they would see

Where is the data? Do we have access to it? This would help to define if the data is in a DB that we can just query or if we need to enable an API for the users to upload data.

Who is going to use the output and how? This helps to define is the model is going to perform batch or stream predictions (i.e. if we need to focus on efficiency) and if the output needs to be stored somewhere for later access or if we should enable an API for its consumption.

What kind of data do we have available? This helps to define if we need a DB (for structured data), a blob storage (for unstructured data) if we need to design that part and if we can cache something if we need low latency.

How many users/predictions should we serve? This helps to define if we need something like microservices or if a monolithic approach should do.

How much data do we have? This helps to define what kind of DB to use and if we should shard (I'm not an expert on DBs, so that's the only way I know to scale).

Do we have access to a training set? This I think would come from the nature of the problem, but it's good to keep in mind.

What are the best metrics to evaluate the model? Idem.

How important is explainability? Idem. Also, this helps to decide/discuss the tradeoffs of black boxes and more traditional approaches.

scaling the DB, cache queries, sharding as an option for scaling the DB, how to address security/privacy (in broad terms since this is not a cybersecurity interview): e.g. keep logs of all requests and fire alerts on certain events, or at least audit them periodically for malicious requests.

Overview

Clarify Requirements
How the ML system fits into the overall product backend
Data Related Activities
Model Related Activities
Scaling
Details

Clarify Requirements
What is the goal? Any secondary goal?
e.g. for CTR - maximizing the number of clicks is the primary goal. A secondary goal might be the quality of the ads/content
Ask questions about the scale of the system - how many users, how much content?
How the ML system fits into the overall product backend
Think/draw a very simple diagram with input/output line between system backend and ML system
Data Related Activities
Data Explore - whats the dataset looks like?
Understand different features and their relationship with the target
- Is the data balanced? If not do you need oversampling/undersampling?
- Is there a missing value (not an issue for tree-based models)
- Is there an unexpected value for one/more data columns? How do you know if its a typo etc. and decide to ignore?

Feature Importance - partial dependency plot, SHAP values, dataschool video (reference)
(ML Pipeline: Data Ingestion) Think of Data ingestion services/storage
(ML Pipeline: Data Preparation) Feature Engineering - encoding categorical features, embedding generation etc.
(ML Pipeline - Data Segregation) Data split - train set, validation set, test set
Model Related Activities
(ML Pipeline - Model Train and Evaluation) Build a simple model (XGBoost or NN)
- How to select a model? Assuming its a Neural Network
    1. NLP/Sequence Model
        - start: LSTM with 2 hidden layers
        - see if 3 layers help,
        - improve: check if Attention based model can help
    2. Image Models - (Don't care right now)
    3. Other

  - start: Fully connected NN with 2 hidden layers
  - Improve: problem specific

(ML Pipeline - Model Train and Evaluation) What are the different hyperparameters (HPO) in the model that you chose and why?
(ML Pipeline - Model Train and Evaluation) Once the simple model is built, do a bias-variance tradeoff, it will give you an idea of overfitting vs underfitting and based on whether overfit or underfit, you need different approaches to make you model better.
Draw the ML pipeline (reference #3)
Model Debug (reference #1)
Model Deployment (reference#3)
(ML Pipeline: Performance Monitoring) Metrics
AUC, F1, MSE, Accuracy, NDCG for ranking problems etc.
When to use which metrics?
Scaling
Be in charge and tradeoffs, tradeoffs, tradeoffs...