ML Breadth Knowledge

lzdh

November 2025

About this document & Acknowledgement

The content of this file come from various sources, mainly generative AI tools and Chip Huyen's (https://huyenchip.com) Machine Learning Interviews book (with deep appreciation). For the LLM Part, most of the information is extracted from Sebastian Raschka's *Ahead of AI* (https://magazine.sebastianraschka.com). This site is brilliant and has so much fun reading about LLM. I highly recommend!

This document was created during the my own interview process and is intended solely for learning and personal study. It may contain mistakes or outdated information, and accuracy is not guaranteed. Please independently verify any technical concepts before relying on them.

This material does not fully reflect the depth and breath of the real ML interviews. Some topics have been expanded with more in-depth content based on the author's personal interests, while others have been intentionally simplified for quicker review.

I am actively updating the document and fixing the errors. If you spot any issues or have suggestions, please feel free to submit a pull request on the author's GitHub repository.

Finally, enjoy reading and enjoy prep! Good luck!

About the author

Personal Website: `https://lzdh.github.io/my_website/`

More interview prep notes can be found on my Rednote (xiaohongshu). Please follow!
Rednote ID: 612410210 `Jessie`

Github: @lzdh
LinkedIn: liu-zdh

Contents

Part I

ML, Stats, ...

Chapter 1

Basics

1.1 Overfitting vs Underfitting

- Overfitting → high variance → sensitive to noise → poor generalization
- Underfitting → high bias → simple assumptions → poor model

1.1.1 Variance–Bias Tradeoff

- Find "sweet point"
- Solution: model selection, hyperparameter tuning
- In clinic, e.g., @ 90% sensitivity / recall

1.1.2 Solution to overfitting

- Data-level: more training data/data augmentation; data normalisation
- Model-level: simpler models; ensemble methods
- Training-level: drop-out; regularization; weight-decay; batch/layer norm; early stopping; cross-validation

1.2 Precision and Recall

$$\text{Precision} = \frac{TP}{TP + FP}$$
$$\text{Recall (Sensitivity)} = \frac{TP}{TP + TN}$$
$$F_1 = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}$$
$$\text{Specificity} = \frac{TN}{N}$$
$$\text{FPR} = \frac{FP}{FP + TN}$$

1.3 Ensemble Methods

1.3.1 Bagging:

- Models trained independently (Parallellizable)
- Training data sampled with replacement (bootstrapping)
- Variance reduction
- Example: Random Forest

1.3.2 Boosting:

- Models trained sequentially
- Focus on previous errors / misclassified samples
- Bias reduction
- Example: XGBoost

1.3.3 Stacking Ensemble

- Combine multiple models by training a "meta-learner" to make final decision
- Stage 1: Base models (diverse algorithms: SVM, logistic regression, etc.)
- Stage 2: Meta-model combines predictions of stage 1 models to make final decision (e.g., linear regression, logistic regression)

1.3.4 Cascading Ensemble

- Stage 1: Simple base model. If confidence is high, make decision. If low, move to Stage 2.
- Stage 2: More complex model. Repeat until confidence is high enough to make decision.

1.4 Decision Tree

1. In classification: each leaf is a class.
2. In regression: each leaf is the mean value of all samples belonging to that leaf. i.e., n leaves $\Rightarrow$ n distinct values. We need Random Forest or XGBoost to generate a wide value range (more continuous-like).
3. Not all features are necessarily used, but all might be used.
4. Splitting:
   - Gini impurity / Cross-Entropy for classification.
   - Variance / MAE / MSE for regression.

1.5 Pruning of Decision Trees

Pre-pruning:

- Early stopping: Max depth; Min cluster size; Min decrease of impurity

Post-pruning:

- Grow full tree, then remove nodes that bring little gain by calculating cost for subtrees.

1.6 Entropy, Gini Impurity, and Cross Entropy

Entropy:

$$H = -\sum_i p_i \log(p_i)$$

Measures uncertainty/randomness of a probability distribution.

Gini Impurity:

$$G = 1 - \sum_i p_i^2$$

Cross Entropy:

$$H(p, q) = -\sum_i p_i \log(q_i)$$

Measures the difference between two probability distributions. Also: $H(p, q) = H(p) + KL(p\|q)$

1.7 Random Forest

1. Bagging: sampling with replacement.
2. At each split, only a random subset of features is used to introduce randomness. This reduces correlation between trees and improves generalisation.

1.8 XGBoost vs LightGBM

1.8.1 XGBoost

Level-wise growth → balanced tree.

- Pros: More stable (especially with small datasets), less prone to overfitting.
- Cons: Slower on large datasets since it considers all splits at each level. May require more splits to reach same loss.

1.8.2 LightGBM

Leaf-wise growth → grows the leaf with the largest potential loss reduction (uneven tree).

- Pros: Lower loss with fewer splits, faster convergence.
- Cons: More prone to overfitting. Requires careful tuning (max depth, min leaf, etc.).

1.8.3 Comparison of Models

	Decision Tree	Random Forest	XGBoost
Structure	Single tree	Bagging / Bootstrapping (Ensemble)	Gradient boosting (Ensemble)
Overfitting	Prone to overfit	Not prone	Avoided by regularization & early stopping
Accuracy	Low accuracy	Better	Best
Interpretability	High	Worse	Worst

1.9 Logistic Regression

$$y = \frac{1}{1 + e^{-(wx+b)}}$$

Sigmoid function. Linear with log-odds.

1.10 Logistic Regression Assumptions

1. Linearity between X and log-odds.
2. Independent observations.
3. No perfect multi-colinearity.

1.11 Linear Regression Assumptions

1. Linearity between X and Y.
2. Independence of errors.
3. Errors have mean 0.
4. Errors have constant variance.
5. No perfect multi-colinearity in features. e.g. $X_3 = \alpha X_1 + \beta X_2$ is perfect multi-colinearity

2,3,4 indicate that **errors are i.i.d.**.

1.12 K-means Assumptions

1. Spherical clusters (round, equally sized).
2. Similar cluster sizes.
3. All features contribute equally (scaling needed).
4. Clusters have uniform density.
5. Convex clusters, separable in Euclidean space.

1.13 Feature Selection

- Filter methods: correlation, chi-square, mutual information.
- Wrapper methods: forward selection, backward elimination.
- Embedded methods: L1/L2 regularization, tree-based methods.

1.14 Generative vs Discriminative Models

Generative:

- Learn the underlying probability distribution of the data X and Y.
- Model $P(X, Y)$ or $P(X|Y)$
- Not only classify, but also generate new data
- Examples: GAN, VAE, Naïve Bayes

Discriminative:

- Learn the decision boundary between different classes
- Model conditional probability $P(Y|X)$
- Examples: SVM, KNN

1.15 KL Divergence

- Measures how a probability distribution differs from a reference probability distribution
- Example: In VAE, acts as a regularization term in the loss. Encourages the learned latent distribution to match a prior (normally standard normal).

1.16 L1 vs L2 Regularization

- L1 (Lasso): drives weights to 0. Geometrically in the loss landscape, it forms diamond contour loss with corners.
- L2 (Ridge): small weights but not 0. Geometrically, the loss landscape is elliptical/circular.
- Elastic Net = L1 + L2

1.17 L2-norm vs Squared L2-norm

- Simpler derivative.
- Smooth and differentiable everywhere, including at 0.
- Numerically stable (no division by 0).
- Squared L2-norm is often used in NN

1.18 Likelihood & Posterior

$$\text{Likelihood: } P(\text{data}|\text{params})$$
$$\text{Posterior: } P(\text{params}|\text{data})$$

1.19 Concept Drift (sys design)

- Relationship between X and Y changes over time

1.20 Semi-Supervised Learning

- Example process:
  1. Train supervised model on labeled data
  2. Predict labels for unlabeled data (select high-confidence labels)
  3. Retrain model including pseudo-labels
- Alternatively: joint/iterative training

$$L = L_{\text{sup}} + \lambda L_{\text{unsup}}$$

where L_{sup} = cross-entropy, L_{unsup} = consistency loss / pseudo-label loss.

1.21 Bayes' Rule

$$P(Y \mid X) = \frac{P(X \mid Y) \cdot P(Y)}{P(X)}$$

- Prior: $P(Y)$
- Likelihood: $P(X \mid Y)$
- Evidence: $P(X)$

1.22 Naïve Bayes

- Assumes all features are conditionally independent given the class label.

$$P(x \mid c = c_i) = \prod_{j=1}^{N} P(x_j \mid c = c_i)$$

- Joint likelihood = product of individual feature likelihoods
- Uses Bayes' rule to compute the posterior of each class given the features
- $P(X, Y) = P(Y) \times P(X|Y)$
- $P(Y \mid X) \propto P(X, Y)$

1.22.1 Naive Bayes Assumptions

All features are conditionally independent given the class.

1.23 AUROC (Receiver Operating Characteristic)

It measures how well a **binary classifier** can distinguish between positive and negative classes. AUROC quantifies a model's ability to rank positives higher than negatives.

$$TPR = \frac{TP}{TP + FN}$$
$$FPR = \frac{FP}{FP + TN}$$

Plot TPR vs FPR.

1.24 R^2 vs Adjusted R^2

- R^2: monotonic, measures amount of variance explained.
- Adjusted R^2: penalizes for unnecessary features, accounts for number of features.

1.25 Correlation and Covariance

Covariance:

$$\text{cov}(X, Y) = \frac{1}{n-1} \sum_{i=1}^{n} (X_i - \bar{X})(Y_i - \bar{Y})$$

Measures how X and Y vary together.
Correlation:

$$\text{corr}(X, Y) = \frac{\text{cov}(X, Y)}{\sigma_X \sigma_Y}$$

Measures the strength of linear relationship between X and Y.

1.26 Maximum Likelihood Estimation (MLE)

- Estimate parameters of a statistical model.
- Assumes data are from a certain distribution with unknown parameters.
- MLE finds the parameter values that maximize the likelihood of the observed data.

$$L(\theta) = \prod_{i=1}^{n} P(x_i \mid \theta)$$

$$\hat{\theta} = \arg\max_{\theta} L(\theta) = \arg\max_{\theta} \sum_{i=1}^{n} \log P(x_i \mid \theta)$$

Often easier to compute log-likelihood (sum instead of product), more numerically stable.

1.27 Class Imbalance Loss

1.27.1 Re-weighted Cross Entropy:

$$L = -w_y \log \frac{e^{z_y}}{\sum_j e^{z_j}}$$

where $w_y = \frac{1}{n_c}$, count in class c.

- Pros: handles imbalance
- Cons: gradient instability for very rare classes, may underfit major classes

1.27.2 Balanced Softmax Cross Entropy (multi-class)

$$L = -\log \frac{n_y e^{z_y}}{\sum_j n_j e^{z_j}}$$

- Rebalances within the softmax
- Effective on long-tail distributions, better than re-weighted CE

1.27.3 Focal Loss (binary, multi-class, multi-label)

$$L = -\alpha(1-p_t)^\gamma \log(p_t)$$

- γ down-weights easy samples; α are class weights
- $\gamma = 0 \Rightarrow$ standard CE
- Larger γ: more imbalance handling (e.g., RetinaNet $\gamma = 2, \alpha = 0.25$ positive, 0.75 negative)

1.27.4 Asymmetric Loss (ASL) (multi-label)

$$L = -y(1-p)^{\gamma^+}\log(p) - (1-y)p^{\gamma^-}\log(1-p)$$

- p: probability from sigmoid, $y \in \{0,1\}$ ground truth
- Typically $\gamma^+ < \gamma^-$, with $\gamma^+ \in [0,1]$, $\gamma^- \in [2,4]$
- Suppresses easy negatives, focuses on hard positives

Comparison:

Focal Loss	ASL
Focus on Hard positives & negatives	Moderate focus on positives, strong suppress negatives
Negatives may dominate loss, positives don't get enough gradient	Works best in multi-label detection with many negatives

1.28 Eigenvectors and Eigenvalues

Given a **square matrix** A, an eigenvector v and its corresponding eigenvalue λ satisfy:

$$Av = \lambda v$$

Eigenvectors show the principal directions of a transformation, and eigenvalues show how strongly those directions are scaled.

1.29 ALS (Alternating Least Squares)

$$M \approx UV^T$$

- Matrix factorization: lower-rank U, V
- Fix U, update V, then fix V, update U:

$$U_i = (V^T V)^{-1} V^T m_i$$

- If $V^T V$ not invertible due to sparsity, use $V^T V + \lambda I$ to ensure positive definiteness

Chapter 2

Neural Network Training

2.1 Batch Norm

- Normalizes activations using statistics computed across the batch dimension.
- Depends on batch size.
- Stabilizes gradient flow
- Allows larger learning rates
- Acts as a form of regularization

2.2 Layer Norm

- Normalizes across hidden dimensions within a single sample, not across the batch
- Independent of batch size
- Work with very small batch, ideal for LLM/Transformers
- Doesn't provide regularization like BatchNorm.

2.3 Exploding Gradient

- Very large gradients cause loss to become NaN or ∞.

Solutions:

1. Gradient clipping: limit the norm of gradients.
2. Weight initialization:

- Xavier for `tanh/sigmoid`.
- He for `ReLU`.

3. Normalization: Batch or Layer normalization.
4. Skip connections: help mitigate, but do not fully solve the problem.

2.4 Vanishing Gradient

- Gradients $\to 0$, causing slow or stalled training.

Solutions:

1. Use ReLU to replace sigmoid/tanh.
2. Skip connections (very effective).
3. Normalization.
4. Proper initialization.

2.5 Gradient Descent

- Iteratively update model parameters to minimize loss

$$\theta_{t+1} = \theta_t - \eta \nabla_\theta J(\theta)$$

where $J(\theta)$ = loss function, ∇_θ = gradient.

2.6 Backpropagation

- Compute gradient of loss function w.r.t each weight parameter using the chain rule
- Propagate errors backward
- Update parameters via gradient descent (negative gradient direction)

2.7 Adagrad

$$\theta_{t+1,i} = \theta_{t,i} - \frac{\eta}{\sqrt{G_{t,i}} + \epsilon} g_{t,i}$$

$$G_{t,i} = \sum_{i=1}^{t} g_{t,i}^2$$

- Each parameter θ has its own adaptive learning rate, scaled by the sum of squared gradients.
- If historical g is large, learning rate decreases; if small, learning rate increases.
- Helps with sparse gradients (e.g., in recommender systems or one-hot features where many parameters are infrequently updated).
- Keeps learning rate high and learn efficiently.

2.8 SGD vs Adam

SGD:

- Fixed learning rate
- Can get stuck in local minima
- 1st order of gradients

Adam:

- Adaptive learning rate based on mean and moments
- 2nd order of gradients

2.9 SGD with Momentum

$$\theta_{t+1} \leftarrow \theta_t - \eta \nabla_\theta L(\theta_t) + \alpha \Delta\theta_{t-1}$$

Momentum (the last term) introduces an exponential moving average of past gradients.

- Converges more slowly compared to Adam.
- Generalizes better.
- Requires tuning of learning rate.

2.10 Adam Optimizer

$$m_\theta^t = \beta_1 m_\theta^{t-1} + (1-\beta_1)\nabla_\theta L^{t-1} \quad \text{(1st order moment)}$$
$$v_\theta^t = \beta_2 v_\theta^{t-1} + (1-\beta_2)(\nabla_\theta L^{t-1})^2 \quad \text{(2nd order moment)}$$
$$\hat{m}_\theta^t = \frac{m_\theta^t}{1-\beta_1^t}, \quad \hat{v}_\theta^t = \frac{v_\theta^t}{1-\beta_2^t}$$
$$\theta^t \leftarrow \theta^{t-1} - \eta \frac{\hat{m}_\theta^t}{\sqrt{\hat{v}_\theta^t} + \epsilon}$$

Typical values: $\beta_1 \approx 0.9$, $\beta_2 \approx 0.999$.

- Combines momentum and adaptive learning rate per parameter.
- Converges faster.
- Generalizes worse (sharp local minima).

2.11 Weight Decay

$$\theta \leftarrow (1-\lambda)\theta - \eta\nabla_\theta L(\theta)$$

- λ is a small decay factor, e.g., $1e^{-4}$.
- Equivalent to L2 regularization.
- Improves generalization.
- Reduces exploding gradients.
- Biases towards simpler solutions.

2.11.1 Weight Decay (Reasoning)

- We prefer small weights in NN training.
- Large weights amplify sensitivity — small perturbations lead to large changes in output.
- Sensitive to noise and prone to overfitting.
- Creates sharp decision boundaries — fits training data well but generalizes poorly.
- If decay factor is too large $(1-\lambda) \to 0$, NN loses learning capacity $\Rightarrow$ underfitting.

2.12 Weight Decay vs L2 Regularization

- In SGD, they are equivalent.
- In Adam, weight decay $\neq$ L2 regularization, since each gradient is rescaled by historical squared gradients.
- Adam has **NO true weight decay**.

2.13 AdamW

Decouples weight decay from gradient update — same principle as SGD.

2.14 Universal Approximation Theorem

Any continuous function can be approximated by a single wide layer NN, but number of neurons may be impractically large.

2.14.1 wide shallow NN vs. deep narrow NN (M)

Certain classes of functions can be represented by deep NN exponentially more compactly than shallow wide NN. Given the same number of neurons, Deep shallow NNs have higher expressiveness and converge faster.

2.15 Saddle Points

At a saddle point, $\nabla L(\theta^*) = 0$, but it is not a local max or min. Hessian has both positive and negative eigenvalues.

- High-dimensional NNs: exponentially more saddle points than local minima.
- Local minima are not problematic in large NNs.
- Saddle points dominate loss landscape $\rightarrow$ main obstacles for gradient descent.
- Optimization gets easier in high dimensions (most local minima are low-loss).
- SGD and momentum help escape saddle points.

Chapter 3

Transformer and Its Derivatives

3.1 Self-Attention

$$\text{Att}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V$$

3.2 Multi-Head Attention

- Multiple self-attention layers in parallel with different learned Q, K, V.
- Allows model to capture information from different representation subspaces.
- Enables focus on different parts of a sequence simultaneously.

3.3 Decoder

1. **Masked multi-head self-attention**
   Uses a causal mask to enforce autoregressive generation. When predicting the next token t, only tokens up to t ($\leq t$) are visible.
2. **Encoder–decoder cross-attention**
   Keys and values come from encoder outputs; queries come from the decoder's previous layer.
   Allows the decoder to condition on the encoded input sequence.
3. **Position-wise feed-forward network (FFN)**

3.3.1 Cross-Attention Notes

- Happens after masked self-attention and before next-token generation.

- The self-attention hidden states serve as queries.
- During training, cross-attention is applied at every decoded token.
- During inference, the decoder computes cross-attention at time t using only the previously generated tokens, and that cross-attention output is used to generate token t.

3.3.2 Self-Attention Notes

- Masked self-attention during training is computed in parallel.
- During inference it is computed sequentially.

3.4 Inductive Bias

- **CNNs**: Locality and translation invariance.
- **RNNs**: Sequential and temporal inductive bias.
- **Transformers**: Global and context-adaptive; Do not assume temporal or local structure; Serve as general-purpose sequence models.

3.5 More Attentions

3.5.1 Grouped-Query Attention (GQA)

- Share key-value heads for multiple query heads.
- Lower memory, improved efficiency.
- Used in Llama, Qwen, GPT, ...

3.5.2 Multi-Head Latent Attention (MLA)

- Compress key & value heads into lower-dim space (before KV cache)
- Extra matrix multiplication, reduces memory usage.
- Used in DeepSeek.

3.5.3 Linear Attention (Performer)

- Approximates softmax using a kernel feature map ϕ.

$$\text{Att}(Q, K, V) \approx \phi(Q)\,(\phi(K)^\top V)$$

- Pros: O(n) complexity.
- Cons: approximation error may accumulate.

3.5.4 Low-Rank Attention (Linformer)

- $QK^\top$ is often approximately low-rank.
- Introduce low-rank projections:

$$K' = KE, \quad V' = VE$$

  where $\text{rank}(K') \ll \text{rank}(K)$.

$$\text{Att}(Q, K, V) = \text{softmax}(Q{K'}^\top)\,V'$$

However, these Linear Attention and Low-Rank Attention mechanisms are rarely used in LLMs due to their approximation errors.

3.6 Attention Collapse

- Attention weights across tokens become uniform or overly concentrated.

3.6.1 Within a single head

Examples:

$$[0.2, 0.2, 0.2, 0.2, \dots] \quad \text{or} \quad [1, 0, 0, 0, 0, \dots].$$

Solution:

- Proper initialization
- Scaled dot-product
- Pre-layer normalization

3.6.2 Across different heads

- Heads learn similar weights or become highly correlated.

Solution:

- Head dropout
- Head diversity regularization

3.7 Activation Functions

3.7.1 GELU (Gaussian Error Linear Unit)

$$\text{GELU}(x) = x \cdot \Phi(x)$$

where $\Phi(x)$ is the CDF of the standard normal distribution.

- Smooth, improves convergence compared to ReLU in deep transformers.
- Used in Transformer, BERT, GPT-2.
- Original activation function in Transformer.
- Computationally more expensive than SiLU, less used now.

3.7.2 Swish (a.k.a. SiLU)

$$f(x) = x \cdot \sigma(x)$$

- Similar to GELU, smoother decay for small negative values.
- Used in EfficientNet, NLP models.

3.7.3 SwiGLU (Swish-Gated Linear Unit)

$$f(x) = (XW_1) \odot \text{Swish}(XW_2)$$

- Combines GLU with Swish (element-wise gating).
- Lets the network learn when to pass signals, improves expressiveness.
- Helps gradient flow and non-linear expressivity.
- Used in LLaMA, PaLM, GPT-4.

3.7.4 GLU (Gated Linear Unit)

$$f(x) = (XW_1) \odot \sigma(XW_2)$$

- Splits into content and gate.
- σ is sigmoid activation.

3.8 Positional Embeddings

- Transformers have no recurrence or convolution, so they need a way to encode order of tokens.
- The (original) positional embeddings (a.k.a. **absolute positional embeddings**) are added to the input embeddings.

3.8.1 RoPE (Rotary Positional Embedding)

- Rotates Q and K vectors relative to their token position.
- Encodes position implicitly by rotating angles.
- No need for extra embedding parameters.
- Lightweight, works better in long sequences.
- Used in LLaMA, Qwen, ChatGLM.

3.8.2 NoPE (No Positional Embedding)

- Typically combined with RoPE for better length generalization.
- Aware of position because of causal attention mask.

3.8.3 ALiBi (Attention with Linear Biases)

$$\text{Attn}(i, j) = \frac{Q_i K_j^T}{\sqrt{d}} + m_h(i - j)$$

- Adds linear bias term depending on relative distance.
- Encourages attention to focus on nearby tokens.
- Simple and cheap computation.
- Used in LLaMA-2, MPT.

3.8.4 Relative Positional Encoding (RPE)

$$\text{Attn}(i,j) = \frac{Q_i K_j^T}{\sqrt{d}} + b_{ij}$$

- b_{ij} = learned embedding/bias depending on relative distance.
- High expressiveness but more expensive to compute.
- Used in Transformer-XL, T5.

3.8.5 Relative RE (Content + Positional Terms)

$$\text{Attn}(Q,K) = QK^T + QR^T + uK^T + vR^T$$

- R encodes relative position.
- Handles long context but memory inefficient.
- Does not generalize well beyond training sequence lengths.

3.9 More Normalizations

3.9.1 RMSNorm (Root mean squared)

$$\text{RMS}(x) = \sqrt{\frac{1}{d}\sum_{i=1}^{d} x_i^2 + \epsilon}$$

$$\text{RMSNorm}(x) = \frac{x}{\text{RMS}(x)} \odot w$$

- w is a scalar
- Doesn't centre data, doesn't compute variance.
- Less computations, more stable.

3.9.2 QKNorm

- Avoids QK^T gets too large (softmax saturates)
- Attention:

$$\text{softmax}\left(\frac{\tilde{Q}\tilde{K}^T}{\tau}\right)V$$

Where:

$$\tilde{Q} = \text{norm}(Q), \qquad \tilde{K} = \text{norm}(K)$$

- $\text{norm}(\cdot)$ could be L2-norm, RMSNorm.
- τ is temperature, a scaling factor.

Chapter 4

Extended Basics

4.1 Norms

Properties of a norm:

1. Non-negativity: $\|u\| \geq 0$, and $\|u\| = 0 \Rightarrow u = 0$
2. Homogeneity: $\|\alpha v\| = |\alpha| \|v\|$
3. Triangle inequality: $\|u + v\| \leq \|u\| + \|v\|$

L_p norm:

$$\|x\|_p = \left(\sum_{i=1}^{n} |x_i|^p \right)^{1/p}$$

Special case: L_0 counts non-zero elements.
Note: Norm $\Rightarrow$ Metric, but Metric $\not\Rightarrow$ Norm.

4.2 Linear Independence

For $c_1 a + c_2 b = 0$, the only solution is $c_1 = c_2 = 0$.

4.3 Convexity / Concavity

- Convex: $f''(x) \geq 0, \ \forall x$
- Concave: $f''(x) \leq 0, \ \forall x$

4.4 First and Second Order Derivatives

Taylor expansion:

$$L(w + \Delta w) \approx L(w) + \nabla L(w)^T \Delta w + \frac{1}{2} \Delta w^T H \Delta w$$

where $H = \nabla^2 L(w)$ is the Hessian matrix.
Update rule:

$$\Delta w = -H^{-1} \nabla L(w)$$

First order ($\nabla L(w)$): set slope
Second order (H): set step size

- Could help with adaptive gradient descent, however due to its memory and computation complexity, rarely used in NN!

4.5 Moments

n-th moment:

$$\mu_n = E[(X - \mu)^n]$$

- 0th: total probability = 1
- 1st: mean
- 2nd: variance
- 3rd: skewness
- 4th: kurtosis

Note: Adam and RMSProp use 2nd order moments, NOT Hessian!

1st order: mean of gradients.
2nd order: variance of gradients, helps smooth noisy gradients, handle parameter scales.

- When 2nd order moment is large ⇒ more uncertainty ⇒ smaller step size.

4.6 Probability Distributions

4.6.1 Normal Distribution

$$X \sim \mathcal{N}(\mu, \sigma^2)$$

$$f(x) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{(x-\mu)^2}{2\sigma^2}\right)$$
$$E[X] = \mu, \quad Var(X) = \sigma^2$$

4.6.2 Binomial Distribution

$$X \sim Bin(n, p)$$

$$P(X = k) = \binom{n}{k} p^k (1-p)^{n-k}, \quad E[X] = np, \quad Var(X) = np(1-p)$$

4.6.3 Multinomial Distribution

$$X \sim Multi(n, \pi)$$

where $\pi = (p_1, p_2, \ldots, p_k),\ \sum p_i = 1$.

$$P(X_1 = x_1, \ldots, X_k = x_k) = \frac{n!}{x_1! \cdots x_k!} \prod p_i^{x_i}, \quad E[X_i] = np_i,\ Var(X_i) = np_i(1-p_i)$$

4.6.4 Poisson Distribution

$$X \sim Poisson(\lambda)$$

$$P(X = k) = \frac{\lambda^k}{k!} e^{-\lambda}, \quad E[X] = \lambda,\ Var(X) = \lambda$$

Poisson $\approx$ Binomial when n large and p small.

4.6.5 Beta Distribution

$$X \sim Beta(\alpha, \beta), \quad \alpha, \beta > 0$$

$$f(x; \alpha, \beta) = \frac{\Gamma(\alpha+\beta)}{\Gamma(\alpha)\Gamma(\beta)} x^{\alpha-1}(1-x)^{\beta-1}, \quad E[X] = \frac{\alpha}{\alpha+\beta},\ Var(X) = \frac{\alpha\beta}{(\alpha+\beta)^2(\alpha+\beta+1)}$$

4.6.6 Exponential Distribution

$$f(x;\lambda) = \lambda e^{-\lambda x}, \quad E[X] = \frac{1}{\lambda},\ Var(X) = \frac{1}{\lambda^2}$$

4.7 Multivariate vs Multimodal Distributions

- Multivariate: multiple variables (e.g., distribution of (X, Y, Z)).
- Multimodal: multiple peaks/modes in PDF (e.g., mixture of two Gaussians).

4.8 PDF, PMF, CDF

- PMF: discrete variable probability at x
- PDF: continuous variable density at x
- CDF: total probability accumulated up to x

4.9 Independence of Random Variables

$$f_{X,Y}(x, y) = f_X(x) f_Y(y)$$

4.10 Frequentist vs Bayesian

	Frequentist	Bayesian
Parameter	Fixed	Random
Probability	Only for data	Both parameter & data
Inference	Point estimate, CI, p-values	Posterior, credible intervals
Prior	None	Yes

4.11 Recommender Systems

4.11.1 Collaborative Filtering

- Recommends items based on user–item interaction history

4.11.2 Assessment

1. **Explicit ratings:** RMSE, MAE, R^2
2. **Implicit feedback:** Precision/Recall@k, NDCG@k, MAP (Mean Average Precision)
3. **Rankings:** NDCG@k, MAP
4. **Business-oriented metrics:** CTR (Click-through rate), Conversion rate

4.12 Active Learning

Model actively selects which data points to label to maximize learning efficiency.

Steps:

1. Train initial model on small labeled set.
2. Model identifies uncertain/informative samples.
3. Query an oracle (human annotator) for labels.
4. Retrain model with new labels → repeat.

Strategies:

- Least confidence: pick samples where top predicted class has lowest probability.
- Margin sampling: pick samples where the difference between top two class probabilities is smallest.
- Entropy-based: pick samples with highest entropy in prediction.

4.13 Weakly Supervised Learning

Labels are incomplete, noisy, bad-quality, or inexact.

- Incomplete label → semi-supervised learning.
- Noisy labels: use robust loss functions.
  - e.g. MAE instead of CE for classification.
  - Label smoothing: soften one-hot labels

$$y_i' = (1 - \epsilon)y_i + \epsilon/k$$

- Generalized CE.
- Sample weighting.
- Soft label loss:

$$y_{target} = \beta y_{noisy} + (1 - \beta) y_{model}$$

4.14 Empirical Risk Minimization (ERM)

Find a model to minimize expected loss on data:

$$R(f) = \mathbb{E}_{(x,y)\sim P_{data}}[L(f(x), y)]$$

$L(\cdot)$ is the loss function, e.g. CE. P_{data} is the true data distribution.

Unknown true data distribution $\rightarrow$ approximate with empirical average (observed data):

$$\hat{R}(f) = \frac{1}{N} \sum_{i=1}^{N} L(f(x_i), y_i)$$

$\Rightarrow$ supervised learning task solved via closed form (if possible) or use NN by gradient descent.

4.15 Parametric vs Non-parametric Models

	Parametric	Non-Parametric
Definition	Fixed form for function/distribution	No assumptions on form
Parameters	Fixed #	Grow with data
Flexibility	Less	More (can approx. a wide range of functio
Computation	Faster	Slower
Examples	Logistic Regression	kNN, k-means, Decision Trees, Gaussian P

4.16 Logistic Regression Loss Function

Use log-loss (cross entropy) for logistic regression, not MSE.

- MSE causes non-convexity when applied to logistic regression.
- Gradient saturation when $\hat{y} \rightarrow 0/1 \Rightarrow$ slow learning.
- CE is convex $\Rightarrow$ global minimum guaranteed.

- CE provides strong gradient when predictions are wrong.
- Theoretically optimal under Bernoulli distribution assumption.
- MSE treats probabilities linearly, but predicted probabilities are log/sigmoid transformed → mismatch.

4.17 K-means vs GMM

- K-means: spherical clusters, hard assignment, non-probabilistic, sensitive to outliers, Lloyd's algorithm.
- GMM: elliptical clusters, soft probabilistic assignments, more robust but computationally heavier, EM algorithm.

4.18 Kernel Methods

1. Linear: $k(x, x') = x^T x'$
2. Polynomial: $k(x, x') = (x^T x' + c)^d$
3. RBF/Gaussian: $k(x, x') = \exp(-\|x - x'\|^2/2\sigma^2)$
4. Sigmoid: $k(x, x') = \tanh(\alpha x^T x' + \epsilon)$

4.19 Reinforcement Learning (RL)

4.19.1 Core Components

- **Agent:** The learner.
- **Environment:** Everything the agent interacts with.
- **State (s):** Description of the environment at time t.
- **Action (a):** Choice the agent makes given s_t.
- **Reward (r):** Feedback signal indicating the benefit of an action.
- **Policy (π):** The strategy the agent uses to make actions based on states.
- **Value function (V):** Estimates how good it is for the agent to be in a state or take an action, in terms of expected future rewards.

4.19.2 Exploration vs Exploitation

Balance between trying new actions vs choosing the best known action.

4.19.3 RL Algorithms

1. **Value-based:** Q-learning
2. **Policy-based:** PPO, REINFORCE
3. **Actor-Critic:** Combines (1) and (2), e.g., A2C, SAC

4.20 TF–IDF

- Term Frequency - Inverse Document Frequency
- Measures word importance in a document

Chapter 5

NLP

5.1 Task Categories

5.1.1 Discriminative Tasks

- NER (Named Entity Recognition)
- NLI (Natural Language Inference)
- QA (Question Answering)
- STS (Semantic Text Similarity)
- IR (Information Retrieval)

Metrics:

- F1-score
- Pearson's correlation
- NDCG@k

5.1.2 Generative Tasks

- Summarization
- Simplification
- Text generation (general)

Metrics: ROUGE,BLEU, METEOR, ...

5.2 Subword Modeling

Reduce vocab size → solve OOV (out of vocabulary) problem. Improve data efficiency.

- **Byte-Pair Encoding (BPE)**: adopted by GPT.
- **WordPiece**: used in BERT.
- **Unigram LM Tokenization / SentencePiece**: used in LLaMA, T5.

5.3 Assessment Metrics

5.3.1 BLEU

- Compares n-gram, computes precision of n-gram, applies brevity penalty (BP).

$$\text{BLEU} = \text{BP} \cdot \exp\left(\sum_{i=1}^{N} w_i \log p_i\right)$$

  p_i is precision.

- Usually computed for unigram to 4-gram.
- **Pros**: Standard, fast, automatic (no human), language-independent.
- **Cons**: Insensitive to semantics, does not capture grammar or fluency.

5.3.2 ROUGE

- Recall-oriented.
- Measures overlap between languages (recall focus, unlike BLEU which is precision).
- **ROUGE-N**: n-gram recall.
- **ROUGE-L**: longest common subsequence.
- **ROUGE-S**: skip-bigram overlap.
- Good to measure summarization.

5.3.3 METEOR

- Considers precision, recall, and alignments such as (Step 1):
  - Exact word match.
  - Stem match (e.g., run $\leftrightarrow$ running).
  - Synonyms.
- Step 2: Computes unigram precision and recall.
- Step 3: Computes F1-score.
- Step 4: Fragmentation penalty (penalize word order differences).

Part II

LLM

Chapter 6

Basics

6.1 Large Language Models (LLMs)

- Generate text autoregressively, outputting probability distributions over tokens.

6.2 Language Model Architecture Categories

- **Encoder-only (bidirectional):** BERT
- **Decoder-only (unidirectional):** GPT, LLaMA, PaLM, ...
- **Encoder–Decoder:** GLM

6.3 LLM Training Pipeline

1. **Pretraining**
2. **Fine-tuning (FT)**
3. **Alignment**

6.3.1 Pre-training

- Masked language modeling
- Next sentence prediction
- Next token prediction

6.3.2 Fine-tuning

1. **Supervised (SFT)**
2. **Instruction tuning (IFT):** instruction–input–output
3. **Parameter-efficient tuning (PEFT)**
   (a) **LoRA** (low-rank adaptation)
   Add trainable low-rank matrices to the self-attention module of each Transformer layer.
   (b) **Prefix:** add a small set of task-specific vectors ("prefix") to the input of each Transformer layer.
   (c) **Adapter:** add small neural network modules ("adapters").

6.4 Prompting (no training)

1. **In-Context Learning (ICL)**
   - One-shot: one example + task description
   - Few-shot: multiple examples
   - Zero-shot: only task description
2. **Chain-of-Thought (CoT)**
   - Generates intermediate steps of reasoning.
   - Can be combined with few-shot prompting.
3. **Prompt tuning**
   - Combines tuning + prompting.
   - Adds learnable prompt vectors.
4. **Retrieval-augmented Generation (RAG)**
   - **Retrieval:** text-embedding-based similarity between queries and documents/knowledge base.
   - **Retrieval types:** adaptive, recursive, iterative.

6.5 Multimodal Models

- Text + Visual
- Examples: **Med-Flamingo, LLaVA-Med, Med-Gemini**

6.6 Autoregressive vs Autoencoding

Autoregressive (e.g., GPT):

- Generate tokens left-to-right.

Autoencoding (e.g., BERT):

- Reconstruct masked tokens.
- Focus on understanding over generation.

6.7 Temperature during inference

- Low temperature ($T \to 0$): peaky probability distribution $\Rightarrow$ more deterministic.
- $T \to 0$ is equivalent to argmax decoding (greedy search).
- $T = 1$: equivalent to normal softmax.
- $T > 1$: more diverse & creative; model becomes more random; probability distribution is flattened.

Softmax with temperature:

$$P_i = \frac{e^{z_i/T}}{\sum_j e^{z_j/T}}. \tag{6.1}$$

6.8 Decoding Strategies

1. **Greedy decoding**: pick the highest-probability token.
   Pros: fast, simple, deterministic.
   Cons: can get stuck in local optima; often yields dull/generic text.
2. **Beam search**: keep k most likely partial sequences at each step; select top-k sequences by cumulative probability.
3. **Sampling-based methods**:

   (a) Random sampling
   (b) Temperature sampling (see above)

4. **Top-k sampling**: sample from the k most probable tokens (common $k = 40$–50).

5. **Top-p (nucleus) sampling**: sample from the smallest set of tokens whose cumulative probability exceeds p.

6. **More advanced**: Contrastive decoding; Typical sampling.

6.9 Modern LLM Decoding

- A combination of processes:
  1. Temperature
  2. Top-p
  3. Repetition penalty
  4. Entropy / typicality constraints
  5. Contrastive (reranking)
- **GPT, Gemini, Claude decoding pipeline:**
  1. Model outputs logits (probabilities)
  2. Apply temperature T
  3. Apply optional penalties, e.g. repetition
  4. Top-p sampling
  5. Post-processing: check for stop sequences, formatting, and safety

6.9.1 Stopping Criteria

1. **Token-based stopping**: detect `<eos>`, `<endoftext>`, or `<s>`; or cut off after a maximum number of tokens.

2. **String-based stopping**: stop when certain strings appear, e.g. `["###End", "\nuser:"]`.

3. **Semantic or logical stopping**.

4. **Safety / alignment-based stopping**.

6.10 Retrieval-Augmented Generation (RAG)

6.10.1 Pipeline

1. User query → embed query → retrieve top-k from vector database.
2. Retrieved chunks are combined with user query to construct prompt → LLM generates response.
3. Benefits:
   - Domain adaptation without fine-tuning
   - Reduced hallucination via factual grounding
   - Increased scalability: expand knowledge base without fine-tuning the LLM

6.10.2 Vector Database (VB)

- Databases optimized for vector similarity search
- Similarity metrics: cosine, dot product, Euclidean distance
- Common vector databases: **FAISS**, **Milvus**, **Qdrant**, **Chroma**

1. **Offline:**
   - Chunking (hundreds–thousands of tokens with overlaps)
   - Embedding generation using embedding models
   - Store metadata (title, date, etc.) with embeddings
   - Index into vector database to enable fast similarity search
2. **Online:** retrieval and query embedding (see pipeline above)
3. **Search algorithms:**
   - ANN (Approximate Nearest Neighbor)
   - HNSW (Hierarchical Navigable Small World Graph): builds layered graph connecting similar embeddings; search by greedy traversal
   - IVF (Inverted File Index): cluster vectors and search only within relevant clusters

Chapter 7

Representative LLMs

7.1 DeepSeek v3 / R1

- **Multi-Head Latent Attention (MLA):**
  - Compresses key/value pairs into lower-dimensional spaces.
  - Works well with KV caching.
  - Shares keys/values among multiple query heads.
- **Mixture of Experts (MoE):** many smaller experts (256); 1 shared expert; fewer active experts (1 to 8 active per step).

7.2 OLMo 2 (Allen Institute)

- Most LLMs use pre-norm (layer normalization before attention).
- OLMo uses post-RMSNorm
- It also adds QK-Norm, an RMSNorm on the query and key inside the attention block, which helps training stability.

7.3 Gemma 3

- Sliding window attention (SWA): SWA:Full attention = 5:1, with window size 1024.
- Both pre- and post-normalization are applied.

7.4 LLaMA 4 (Similar to DeepSeek v3)

- Grouped Query Attention (GQA)
- Larger but fewer MoE experts; alternates MoE layers with dense layers.

7.5 Qwen 3

- Deeper but narrower than LLaMA 1B.
- No shared MoE.

7.6 SmolLM 3

- NoPE every fourth layer with RoPE for positional encoding.

7.7 Kimi 2

- Uses Muon optimizer instead of AdamW.
- Employs reinforcement learning (RL) fine-tuning.

7.8 GPT-OSS (Open Weight GPT)

- MoE: 32 experts, 4 active per forward pass.
- RMSNorm and GQA with RoPE positional encoding.
- SiLU / SwiGLU activation functions.
- No dropout.
- Sliding window attention.
- Each attention head uses a learned bias in the softmax denominator — effectively a per-head 'attention sink' mechanism
- **Attention sinks:** always-attended tokens at the start of sequence to stabilize attention.

7.9 GPT-OSS vs GPT-2 (2019)

Feature	GPT-OSS (2025)	GPT-2 (2019)
MoE	4 active experts w/ SiLU/SwiGLU	Single FF with GELU
Positional Encoding	RoPE	Absolute PE
Normalization	RMSNorm	LayerNorm
Attention	GQA + Sliding Window	Self-Attention
Dropout	None	Applied
Embedding Dim	2880	1600
Context Length	131k	1024
Attention Heads	64	25
Transformer Layers	24	48
Vocab Size	200k	50k

7.9.1 Design Notes

- Modern LLMs generally omit dropout.
- SiLU/SwiGLU preferred over GELU (slightly cheaper, more efficient).
- In MoE models, expert weights account for $> 90\%$ of total parameters!!

7.9.2 Core Techniques

1. **GQA:** groups heads to share key/value to lower parameter count and reduce memory bandwidth.
2. **Sliding Window Attention:** applied in GPT-OSS in every second layer, reducing compute and memory usage.
3. **MoE:** experts are feed-forward modules with fully connected layers and activation functions.
4. **GPT-OSS Reasoning:** controllable reasoning levels (low/medium/high) based on inference-time scaling.
5. **MXFP4 Quantization:** enables MoE models to run efficiently on a single GPU:
    - 120B on one H100 (80GB)
    - 20B on RTX 50-series (16GB)
    - Without MXFP4: 20B consumes 48GB RAM; 120B consumes 240GB.

7.10 DeepSeek RL Reasoning

- DS-R1-Zero: pure RL (RLVR)
- DS-R1: SFT + RL (RLVR + RLHF)
- DS-Distill: SFT only

R1-Zero: rule-based reward system

- Accuracy reward
- Format reward

RLVR induces reasoning; pretraining on CoT data also improves reasoning ability.

7.11 More Notes on Reasoning

1. Kimi RL-S uses a Process Reward Model (PRM), similar to RLVR but with verifiable rewards.
2. PRM evaluates both final answers and reasoning steps using:
   - Accuracy reward
   - Log-length reward
3. Small LLM reasoning:
   - Distillation better than pure RL for small LLMs.
   - RL on distilled small LLM improves performance.
   - Training for 50–100k steps with high-quality data improves performance before plateau.
4. Some later study ('sober look') suggests these improvements may be partially due to noise.

Chapter 8

Reasoning

8.1 Recap LLM Training Pipeline

1. **Pretraining:** MLM / next-token or sentence prediction.
2. **Supervised fine-tuning (SFT):** next-token prediction on instruction–output pairs (human-generated, smaller dataset).
3. **Alignment:** human preference alignment (e.g. RLHF with PPO), typically $\sim$50k samples.

Reward Model (RM): generally originates from the same LLM, with the last layer replaced by a regression head.

8.2 Reinforcement Learning from Human Feedback (RLHF)

8.2.1 ChatGPT

1. **Stage 1:** SFT (Supervised Fine-Tuning) of the pretrained model.
2. **Stage 2:** Create a reward model.
3. **Stage 3:** Fine-tune via Proximal Policy Optimization (PPO).

Stage 2 details:

- Gather 4–9 responses after S1.
- Humans rank responses (time & labor intensive).
- Train a reward model (another LLM) — replace next-token prediction head with a regression layer.

Stage 3: Update the fine-tuned model using PPO based on reward scores from S2.

8.2.2 LLaMA-2)

- Two reward models: helpfulness and safety.
- Generate 2 responses (instead of 4–9); adopt margin label/loss (e.g. significantly better vs. slightly better).
- Iteratively produce multiple RLHF models using:
  - Rejection sampling (Step 1 & Step 2)
  - PPO (Step 2)

8.3 Proximal Policy Optimization (PPO)

- **Policy:** the LLM being optimized.
- Limit the change of policy per update step to avoid destabilization.
- Add a KL-divergence penalty to the loss.
- Add entropy bonus to encourage exploration.

8.3.1 Computation Steps

1. Compute ratio = new policy prob / reference policy prob.
2. Raw score = ratio × advantage
   Advantage = difference between actual reward and expected reward.
3. Expected reward computed by a **critic model** (value model) via Generalized Advantage Estimation (GAE).
4. If policy changes too much (e.g., ratio > 1.2 or < 0.8), clip ratio.
   Clipped score = clipped ratio × advantage.
5. If advantage > 0: final score = min(raw, clipped)
   If advantage < 0: final score = max(raw, clipped)
6. Calculate loss:

$$L = -(\text{final score}) + \beta \times D_{KL}(\text{new policy} \| \text{ref policy}) \tag{8.1}$$

8.3.2 Models Involved in PPO

1. Policy model — LLM being optimized.
2. Reward model — often the same LLM but fine-tuned.
3. Critic model — trainable, estimates reward (value model).
4. Reference model — frozen copy of the original LLM.

Reward model remains frozen; critic continues updating.

8.4 GRPO (Group Relative Policy Optimization) — DeepSeek

- Removes the critic/value model.
- Samples multiple answers from the policy; computes average reward as the expected reward to estimate advantage.

8.5 RLHF → RLVR (Verifiable Reward) — DeepSeek

- Replaces the reward model with verifiers or rule-based tools.
- RLVR with GRPO: no critic, no reward model.
- Effective for tasks like mathematics and coding.

8.6 Alternatives to RLHF

8.6.1 Constitutional AI (CAI, Dec 2022):

- Aligns LLM using AI feedback guided by human-defined rules/principles/constitutions.
- Stages:
  1. Supervised self-training: pretrained model follows rules → generates responses → self-critiques → revises.
  2. RLAIF: model ranks responses according to same rules and fine-tunes via RL.
- CAI is an alignment framework.

8.6.2 RLAIF (Sep 2023):

Ratings for reward models generated by AI (second stage of CAI).

8.6.3 Directed Preference Optimization (DPO, May 2023):

- Alternative to RLHF with PPO — more efficient.
- Does not require reinforcement learning; directly optimizes the policy (LLM) to prefer desired outputs using a sigmoid Cross-entropy loss.
- Cross-entropy loss over logits of preferred vs. less good outputs.
- Procedure:
  1. Have two candidate responses (chosen/rejected).
  2. Compute log probabilities under current model and reference model (frozen copy).
  3. Compute DPO loss (sigmoid CE form).
  4. Back-propagate.

Supervised-like fine-tuning only.

8.6.4 Contrastive Preference Learning (CPL, Aug 2023):

- Similar to DPO; no RL, no reward model.
- Supervised fine-tuning with contrastive loss.

8.6.5 Reinforced Self-Training (ReST):

- Sampling-based iterative creation for better fine-tuning dataset generation.
- fine-tuning on this dataset.

8.7 Recent LLM Reasoning Models (2025)

8.7.1 General Strategy

1. **Inference-time scaling (test-time compute)**.
2. **Pure RL (train-time compute).**
3. **RL + SFT (train-time compute).**
4. **Pure SFT / Distillation (train-time compute).**

8.7.2 Strategies to Improve Reasoning

- Increase training compute.
- Increase inference compute (inference/test-time scaling (ITS)).
- Combine the two.

8.7.3 Classic ITS Approaches

Prompt Engineering Approaches

- Chain-of-Thought (CoT), RAG, etc.
- CoT generates longer responses — more expensive for simple tasks.

Search Strategies

- Best-of-N, beam search, lookahead search.

8.7.4 Recent ITS Techniques

1. **Simple Test-Time Scaling (TTS):**
   - Requires SFT on few examples ($\approx$ 1k samples) with training traces .
   - Control response length by :
     - appending "**wait**" token to lengthen response, for self-correct
     - using "Find Answer" token to stop generation.
2. **Test-Time Preference Optimization (TPO):**
   - Generate multiple responses per prompt.
   - Score with reward model.
   - Prompt model to compare best vs. worst.
   - Refine output by incorporate these critiques.
   - iterate previous steps.
3. **Thought Switching Penalty (TIP):**
   - Address 'underthinking' issue.
   - Penalize premature reasoning path switching by modify logits of thought-switching tokens.
4. **Chain-of-Associated-Thought (GoAT):**

- Monte Carlo Tree Search + Associative Memory (knowledge base).
- During exploration of reasoning paths, dynamically incorporate reasoning information for response generation.

5. **Step Back to Leap Forward:**
   - Self-backtracking: learn when and where to backtrack during training and inference for correction.
   - Train the model to recognize and correct suboptimal paths using a <backtrack> token.
   - Tree-based search at inference; doesn't rely on external reward model.
6. **Compute-Optimal Test-Time Scaling:**
   - Most TTS reply on sampling, which requires Process Reward Model (PRM) to select best answers.
   - Proposed a compute-optimized scaling strategy adapts PRM, policy model and tasks to maximize efficiency.
   - Smaller models with proper TTS can outperform larger ones.
7. **Chain-of-Draft (CoD):**
   - Generate fewer tokens; improves TTS efficiency via structured prompting.

8.7.5 Conclusions

- No single method wins universally.
- Simple token-based $\rightarrow$ search & optimization-based methods.
- Small LLMs + substantial TTS $\Rightarrow$ competitive with larger LLMs.
- TTS is becoming a standard technique like RLHF.

Chapter 9

Evaluation

9.1 General Overview

1. **Standard NLP Benchmarks:**
   - MMLU (Knowledge & Reasoning)
   - BBH
   - ARC (AI2 Reasoning Challenge)
2. **Commonsense QA**
3. **Text Generation:**
   - **Fluency:** Measured by perplexity (lower is better)
   - **Relevance:** Cosine similarity, BERTScore
   - **N-gram overlap:** BLEU, METEOR, ROUGE, F1 score
4. **Human Evaluation / Qualitative:**
   (a) Expert review
   (b) Pairwise comparison (Likert scale)
5. **Task-specific Evaluation**
6. **Efficiency Metrics:** Latency, cost

9.2 LLM Evaluation

- **Benchmark-based:**
  1. Multiple-choice benchmarks (e.g., MMLU with 57 subjects).

2. Verifiers (quantify LLM accuracy using external tools).

- **Judgment-based:**

1. Leaderboards.
2. LLM judges (automated evaluation via strong models).

9.2.1 Multiple Choice Evaluation

1. MMLU: 57 subjects $\approx$ 16k questions; measures accuracy across broad domains.
2. External tools: calculators, code interpreters for reasoning tasks.
3. LM Arena: pairwise user preference comparison between models.
4. Automated evaluation (no human feedback): use strong LLMs as judges with predefined rubrics via API (e.g., ChatGPT, Claude).

9.2.2 Verifiers

- Designed for free-text output evaluation.
- Quantify LLM capabilities via accuracy metrics.
- Use external tools (calculators, code execution) for factual verification.

Chapter 10

LLM Optimization

10.1 Optimization Techniques

10.1.1 Quantization

- Reduce numerical precision of model parameters to improve inference speed and save memory.
- **Post-training quantization (PTQ):** quantize pretrained model without retraining (slight accuracy drop).
- **Quantization-aware training (QAT):** simulate quantization during fine-tuning for better accuracy.
- **Mixed-precision inference:** use FP16 for sensitive layers (e.g., attention softmax), INT8 for others.

10.1.2 MXFP4 Quantization (GPT-OSS)

- Microscaling FP4-bit representation: 1 sign bit, 2 exponent bits, 1 mantissa bit (E2M1) + Block-level scaling factors.
    1. Tensors are divided into groups (e.g., 32 values per block). Each block has a shared scale factor to map the quantized 4-bit back to float values.
    2. Encoding: store each value in E2M1 format.
    3. Two FP4 values are packed in one 8-bit unit memory.
    4. MoE weights quantized with MXFP4, other layers stay at higher precision.

10.1.3 SmoothQuant

- 8-bit quantization for both weights and activations (W8A8).

- Low precision and memory drop; no retraining required (PTQ).

10.1.4 AWQ (Activation-Aware Weight Quantization)

Typically uses 4-bit weight quantization (W4).

- Protects salient weights based on activation statistics.
- No backpropagation needed (requires a calibration set).
- Provides lower bandwidth and better compression efficiency.

10.1.5 GPTQ (Gradient-based PTQ)

- Quantizes weights layer-by-layer using Hessian approximation to minimize quantization error.
- No training required; uses a small calibration dataset.
- Precision typically ranges from 2–8 bits.

10.1.6 QLoRA (Quantized Low-Rank Adapter)

- Takes a 4-bit quantized LLM and injects LoRA adapters inside Transformer blocks.
- Quantized weights remain frozen; only LoRA parameters (e.g., 16-bit) are trainable.

10.2 Token Pruning

- Removes less important tokens during inference to reduce computation.
- Risk: pruning important tokens causes context loss.

1. Compute importance scores (e.g., attention weights, learned gating).
2. Mask or remove low-importance tokens dynamically or statically.
3. Model only attends the remaining tokens.

10.2.1 Pruning Strategies

1. Dynamic pruning: model decides which tokens to keep during inference.
2. Static pruning: certain tokens or positions are always pruned.
3. Layer-wise pruning: more aggressive pruning in deeper layers.

LazyLLM: dynamically select important tokens to compute; pruned tokens can be revived if becoming relevant later.

10.3 LLM Agents and Multi-Agent Systems

10.3.1 Definition

An **agent** is an LLM with:

- State (knowledge or memory)
- Policy (decision-making rules)
- Action and goal

10.3.2 Multi-Agent Systems

- Roles: Planner, Solver, Evaluator, Coordinator.
- Multi-agent setups enable structured reasoning and cross-verification to reduce hallucinations.

10.3.3 Example Pipeline

1. **Planner:** breaks queries into sub-problems.
2. **Retriever:** fetches external knowledge or tools.
3. **Solver:** generates draft output.
4. **Critic / Verifier:** checks factual consistency and logical correctness.
5. **Aggregator / Judge:** consolidates verified results.

Part III

System Design

Chapter 11

When to Trigger Model Refresh

11.1 Performance-based

1. **Metric degradation:** AUROC, Accuracy, F1, RMSE, etc.
2. **Calibration drift:** predicted probabilities no longer align with real-world frequencies.
   Detect by plotting predicted probability vs. observed frequency.
3. **Brier score (binary class):**

$$\text{Brier} = \frac{1}{N}\sum_{i=1}^{N}(p_i - y_i)^2 \tag{11.1}$$

4. **Expected Calibration Error (ECE):**

$$\text{ECE} = \sum_{m=1}^{M}\frac{|B_m|}{N}\Big|\text{acc}(B_m) - \text{conf}(B_m)\Big| \tag{11.2}$$

5. **Segmented performance:** look for drops in certain user groups / slices.

11.2 Data-based

1. **Data distribution shift:** e.g., KL divergence growth.
2. **Concept drift:** $X \not\Rightarrow Y$ due to regulation changes or new behaviors.
3. **Data quality:** rising missingness, noise, or label issues.

11.3 Hybrid

Regular cadence retraining + trigger-based refresh based on the above criteria.

Chapter 12

Design a Bad User Detection System

12.1 Problem Framing

- Clarify problem type: binary, multiclass, or regression.
- Quantify costs of False Positives (FP) and False Negatives (FN).
- Identify interpretability needs and real-time vs. queued review constraints.
- Example: for fraud detection, define goals like "reduce chargebacks" or "flag high-risk transactions for manual review."

12.2 Data & Labeling

- Create a data labeling pipeline including:
  - Label values (e.g., fraud/not-fraud)
  - Confidence or reviewer agreement
  - Timestamp of labeling
- Typical features: IP address, payment method, shipping address, time-of-day, historical risk, graph links (shared IP or device).

12.3 Modeling Choice

1. **Strong baseline:** XGBoost (for tabular data).
2. Try simpler models (e.g., logistic regression) to test linearity, interpretability, and speed.
3. When sequential patterns matter, use RNNs, Transformers, or temporal CNNs.

4. Combine offline and online modeling:
    - Offline trains a foundation embedding per buyer.
    - Online combines embeddings with real-time signals (payment method, IP) and feeds to XGBoost for scoring.

12.4 Advanced Modeling Techniques

- **Graph models:** for collusion detection (shared accounts, addresses, IPs, etc.).
- **Unsupervised learning:** for anomaly and novel pattern detection.
- **Handling class imbalance:**
    - Weighted class loss (Focal Loss, ASL) or class weights.
    - Resampling (e.g., SMOTE for upsampling).
    - Ensemble models on balanced subsets.
    - Two-stage modeling:
        1. High-recall lightweight model to surface candidates (e.g., trained on downsampled data).
        2. More powerful model to refine and increase precision.

12.5 Evaluation & Thresholding

- Metrics:
    - AUPRC, F1, Precision@k, Recall at fixed FP, cost-based expected value.
- Cross-validation:
    - Use time-based splits for temporal data.
    - Ensure test sets are representative.
- Thresholding:
    - Choose thresholds that maximize expected utility or cost-adjusted performance.

12.6 Production Monitoring

- Conduct A/B tests to measure:
  - Fraud reduction
  - Customer experience metrics
- Monitor:
  - Latency
  - Calibration
  - Model drift
- Include **human-in-the-loop** for review and ongoing refinement.

Chapter 13

Design a LLM-based Summarization System

13.1 Questions to Ask the Interviewer

Data sources (Word, PPT, emails), summary length/format (bullets vs. paragraphs), users (multilingual/external), and evaluation metrics (ROUGE, BLEU, human eval).

13.2 Pipeline

1. User document → *Preprocess* → *Embed/Encode* → *Summarize*
2. Model output → *Post-process* → *Display*

13.3 Preprocess

- Tokenization, text cleaning; handle tables/images.
- Tables → structured data or natural language; images → ViT or captioner.
- Text: remove only unnecessary tokens; normalize text.
- If documents are long: chunking with overlap; later merge chunks logically.

13.4 Embedding

Use a pretrained tokenizer/encoder (e.g., BERT/Longformer).

13.5 Summarization Model

Prefer a pretrained & fine-tuned LLM for summarization.

13.6 Post-process

Grammar/coherence checks; text cleaning (remove unwanted tokens/duplicates); convert to user-desired format; allow external tools or LLMs to assist formatting.

13.7 Feedback Loop

Collect signals for continuous learning & evaluation:

- User satisfaction / accept rate
- Time saved (self-reported)
- Model latency
- Engagement (usage frequency)
- Error/correction rate (edits)

13.8 Model Generation (Fine-tuning an LLM)

13.9 Data Preparation (for SFT)

Collect input-output pairs; outputs may be human/machine-generated reference summaries; chunking if necessary.

13.10 Model Selection

Pick base model considering latency, context length, multilingual needs, privacy, and licensing.

13.11 Fine-tuning Approaches

1. Full SFT
2. Parameter-Efficient FT (PEFT): e.g., LoRA adapters, prefix-tuning
3. (Optional) RLHF / alignment

13.12 Training Objective

- Cross-Entropy loss with teacher forcing:

$$\mathcal{L}_{CE} = -\sum_{t} \log P(y_t^* \mid y_{<t}, X) \tag{13.1}$$

- Auxiliary losses: coverage loss; contrastive loss if multimodal alignment is required.

13.13 Evaluation

BLEU, ROUGE, METEOR, plus human evaluation.

13.14 Inference / Generation

- Decoding strategies: beam search, top-k, top-p with temperature.
- Post-processing and formatting.