

263-5354-00: Large Language Models

Conceptual Summary

Swiss Federal Institute of Technology Zurich (ETH Zürich)

Last Edit Date: 09/09/2025

Disclaimer and Term of Use:

1. We do not guarantee the accuracy and completeness of the summary content. Some of the course material may not be included, and some of the content in the summary may not be correct. You should use this file properly and legally. We are not responsible for any results from using this file.
2. This personal note is adapted (derived) from the slides and codes from *Professor Ryan Cotterell, Mrinmaya Sachan, and Florian Tramèr*. Please [contact us](#) to delete this file if you think your rights have been violated.
3. This work is licensed under a [Creative Commons Attribution 4.0 International License](#).

1. Language Model Definitions

1. Set of strings

An alphabet Σ is a finite, non-empty set. A **string** over an alphabet is any finite sequence of letters. The **Kleene star** $\Sigma^* = \bigcup_{n=0}^{\infty} \Sigma^n$, where $\Sigma^n = \Sigma \times \dots \times \Sigma$. We define $\Sigma^0 = \{\varepsilon\}$. A **language** L is a subset of Σ^* .

II. Language model

Alphabet Σ , a **language model** is a (discrete) distribution p_{LM} over Σ^* . The **weighted language** of p_{LM} is $L(p_{LM}) = \{(y, p_{LM}(y)) | y \in \Sigma^*\}$.

III. Sequence model

Alphabet Σ , a **sequence model** is defined as a set of conditional probability distributions $p_{SM}(y|y_{<t})$, for $y \in \Sigma$ and $y \in \Sigma^*$ is the context.

2. Global and Local Normalization

I. Globally normalized language models

An **energy function** is a function $\hat{p}: \Sigma^* \rightarrow R$. Let $\hat{p}_{GN}(y): \Sigma^* \rightarrow R$ be an energy function. A **globally normalized model** is defined as

$$p_{LM}(y) = \frac{e^{-\hat{p}_{GN}(y)}}{\sum_{y' \in \Sigma^*} e^{-\hat{p}_{GN}(y')}} = \frac{1}{Z_G} e^{-\hat{p}_{GN}(y)}, Z_G = \sum_{y' \in \Sigma^*} e^{-\hat{p}_{GN}(y')}$$

$\sum_{y' \in \Sigma^*} e^{-\hat{p}_{GN}(y')}$ is the normalization constant.

Theorem: Normalizable energy function p_{GN} induces a language model. Given $e^{-\hat{p}_{GN}(y)} \geq 0$, we get $\sum_{y \in \Sigma^*} p_{GN}(y) = \sum_{y \in \Sigma^*} \frac{e^{-\hat{p}_{GN}(y)}}{\sum_{y' \in \Sigma^*} e^{-\hat{p}_{GN}(y')}} = 1$.

$$\frac{1}{\sum_{y' \in \Sigma^*} e^{-\hat{p}_{GN}(y')}} \sum_{y \in \Sigma^*} e^{-\hat{p}_{GN}(y)} = 1.$$

Remark: Energy function is normalizable if $Z_G < \infty$. Computing Z_G efficiently is challenging, as Σ^* is infinite, Z_G is not tractable, difficult to sample from.

II. Locally normalized language models

Define $\bar{\Sigma} = \Sigma \cup \{EOS\}$, denote elements of $\bar{\Sigma}^*$ as $\bar{y}$, symbols in $\bar{\Sigma}$ as $\bar{y}$. Alphabet Σ , let p_{SM} be a sequence model over $\bar{\Sigma}$, a locally normalized language model over Σ is $p_{LN}(y) = p_{SM}(EOS|y) \prod_{t=1}^n p_{SM}(y_t|y_{<t})$, where $y \in \Sigma^*$ with $|y| = T$.

Theorem: Any language model can be locally normalized.

Remark:

- A locally normalized language model is **tight** if $\sum_{y \in \Sigma^*} p_{LN}(y) = 1$.

III. Prefix probability

p_{LM} is a language model. p_{LM} 's prefix probability is $\pi(y) = \sum_{y' \in \Sigma^*} p_{LM}(yy')$, which is the probability that y is a prefix of any string yy' in the language. $\Sigma = \{I, \text{like}, \text{dogs}\}$, $P(I) = 0.5$, $P(\text{like}|I) = 0.6$, $P(\text{dogs}|I \text{ like}) = 0.8$, then $P(I \text{ like dogs}) = 0.5 \times 0.6 \times 0.8$.

Remark: Naturally, $\pi(\varepsilon) = 1$.

3. Tightness

A locally normalized language model p_{LN} derived from a sequence model p_{SM} is called **tight** if $\sum_{y \in \Sigma^*} p_{LN}(y) = \sum_{y \in \Sigma^*} [p_{SM}(EOS|y) \prod_{t=1}^n p_{SM}(y_t|y_{<t})] = 1$.

An LNM is tight if and only if $\hat{p}_{EOS}(t) = 1$ for some t or $\sum_{t=1}^{\infty} \hat{p}_{EOS}(t) = \infty$.

4. Modeling Foundations

I. Representation-based LNM

A representation-based locally normalized model is $p_{SM}(y_t|y_{<t}) = f_{\Delta|\Sigma|+1}(Enc(y_{<t}))_{y_t}$,

we assume $f_{\Delta|\Sigma|+1}$ to be softmax unless otherwise stated. It defines the probability of an entire string $y \in \Sigma^*$ as $p_{LN}(y) = p_{SM}(EOS|y) \prod_{t=1}^n p_{SM}(y_t|y_{<t})$, where $y_0 = BOS$.

Remark: A locally-normalized representation-based language model with uniformly bounded $\|enc(y)\|_p$ (for some $p \geq 1$) is tight.

II. Estimating a LM from data

We search for model parameters $\hat{\theta} \in \Theta$ such that the model maximizes a chosen objective or minimize loss, $\hat{\theta} = \argmin_{\theta} \ell(\theta^*, \theta)$. We seek a model p_{θ} that is close to p_{θ^*} . Common losses include cross entropy, KL divergence. Maximum likelihood is usually used, $\hat{\theta}_{MLE} = \argmax_{\theta} L(\theta)$, where $L(\theta) = \prod_{n=1}^N p_{\theta}(y^{(n)})$, which maximize the likelihood of observing the given data under that model. Typically, it works with the log-likelihood $\log L(\theta)$.

III. Softmax

Let $\tau \in R_+$ be the temperature. The softmax at τ is the projection function $softmax(x)_d = \frac{e^{x_d/\tau}}{\sum_{i=1}^D e^{x_i/\tau}}$, for $d = 1, \dots, D$. High τ produces uniform probability distribution, low τ produces peaked probability distribution.

Remark (desired properties of softmax): - The softmax is invariant to adding the same constant to all coordinates in x .

- The derivative of the softmax is continuous and differentiable everywhere, optimization.

- The softmax maintains the rank of x .

5. Finite State Language Models

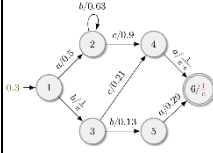
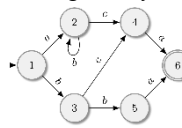
A language model p_{LM} is **finite-state** if it defines only finitely many unique conditional distributions $p_{LM}(y|y)$.

I. Finite-state automata (FSA)

A FSA is a 5-tuple $(\Sigma, Q, I, F, \delta)$ where Σ is an alphabet, Q is a finite set of states, $I \subseteq Q$ is the set of initial states, $F \subseteq Q$ is the set of final or accepting states. A finite multiset $\delta \subseteq Q \times (\Sigma \cup \{\varepsilon\}) \times Q$. Elements of δ are generally called **transitions**.

Remark: A FSA $A = (\Sigma, Q, I, F, \delta)$ is **deterministic** if it does not have any ε -transitions, for every $(q, a) \in Q \times \Sigma$, there is at most one $q' \in Q$ such that $q \xrightarrow{a} q' \in \delta$, there is a single initial state $|I| = 1$. Otherwise, A is **non-deterministic**.

II. Real-weighted FSA (WFSA)



Q a finite multiset of transitions, $\lambda: Q \rightarrow R$ a weighting function over Q , $\rho: Q \rightarrow R$ a weighing function over Q . **String probability** $p_A(y) = \frac{A(y)}{Z(A)}$, where $y \in \Sigma^*$ is a string, where $Z(A) = \sum \lambda M \rho$, M is the sum of all inner weights for all paths.

III. Probabilistic weighted FSA (PFSA)

A WFSA $A = (\Sigma, Q, \delta, \lambda, \rho)$ is a PFSA if $\sum_{q \in Q} \lambda(q) = 1$ and for all $q \in Q$ and all outgoing transitions $q \xrightarrow{a/w} q' \in \delta$ it holds $\lambda(q) \geq 0$, $\rho(q) \geq 0$, $w \geq 0$ and $\sum_{q \rightarrow q'} w + \rho(q) = 1$.

Remark: The final weights in a PFSA play an analogous role to the EOS symbol.

IV. LN and GN finite-state language models

Theorem: Normalizable WFSAs and tight PFSA are equally expressive. **New weights:** initial: $\lambda_{A_L}(q) = \lambda(a) \frac{Z(A,q)}{Z(A)}$, final: $\rho_{A_L}(q) = \frac{\rho(q)}{Z(A,q)}$,

$$\text{transition: } w_{A_L}(q \xrightarrow{a/w} q') = \frac{w(q \xrightarrow{a/w} q') Z(A,q')}{Z(A,q)}.$$

6. N-gram Models

Assumption: Conditional probability of the symbol y_t given $y_{<t}$ only depends on $n - 1$ previous symbols $p_{SM}(y_t|y_{<t}) = p_{SM}(y_t|y_{t-n+1}^{t-1})$, $y_{t-n+1}^{t-1} = y_{t-n+1}, \dots, y_{t-1}$. **N-gram models** are defined in terms of local normalization. Consider tight sequence model $\{p(\cdot|x)\}_{x \in \Sigma^*}$, $p(\cdot|x)$ is over $\Sigma \cup \{EOS\}$. $\{p(\cdot|x)\}_{x \in \Sigma^*}$ is an **n-gram model** if $p(\cdot|x) = p(\cdot|x_{t-n+1} \dots x_t)$, having $n - 1$ symbols.

Parameterization: $\theta = \{\theta_{y|y'} = p_{SM}(y|y') | y \in \Sigma, y' \in \Sigma^{n-1}, \theta_{y|y'} \geq 0, \sum_{y' \in \Sigma^{n-1}} \theta_{y|y'} = 1\}$. **MLE**

solution: $p_{SM}(y_n|y_{<n}) = \frac{C(y_1, \dots, y_{n-1})}{C(y_1, \dots, y_{n-1})}$, $C(\cdot)$ denotes the occurrences of all possible strings of the form in the bracket.

Remark: $\{p(\cdot|x)\}_{x \in \Sigma^*}$ is countably infinite distribution. There are $O(|\Sigma|^n)$ distributions. **Smoothing** – sharing parameters among the $p(\cdot|x)$'s. **N-gram LMs are finite-state LMs.**

7. Recurrent Neural Language Models

I. RNN definition and property

A **deterministic real-valued RNN** is a 4-tuple (Σ, D, f, h_0) , where Σ is alphabet, D is the dimension of hidden state space, $f: R^D \times \Sigma \rightarrow R^D$ is the transition function (updates hidden state), $h_0 \in R^D$ is the initial state vector.

Remark: A softmax RNN is tight if the norm of its hidden states doesn't grow too fast: $s\|h_t\|_2 \leq \log t$. RNNs with **unbounded activations** are not guaranteed to be tight.

II. Elman RNNs

An Elman SM, $R = (\Sigma, D, U, V, E, b_h, h_0)$ is a D-dimensional RNN sequence model over an alphabet Σ with the following dynamics map $h_t = \sigma(Uh_{t-1} + Ve'(y_t) + b_h)$. $e': \Sigma \rightarrow R^R$ is the input symbol embedding function, which represents each symbol $y \in \Sigma$ as a R-dimensional vector and σ is an element-wise non-linearity. $b_h \in R^D$, $U \in R^{D \times D}$, and $V \in R^{D \times R}$.

III. Heaviside Elman Network (HRNN)

Theorem: An Elman RNN that uses the Heaviside activation function is computationally equivalent to a deterministic PFSA. Non-differentiable activation - **Heaviside function** $H(x) = 1$ if $x > 0$, $H(x) = 0$ otherwise, forcing hidden states of RNN to be a vector of only 0s and 1s. An RNN is **no more powerful than a dPFSA**, due to $H(x)$, there are only 2^D possible hidden states. An HRNN can **simulate any dPFSA** as its hidden state h_t is a one-hot vector, the dimension is $D = |Q| \times |\Sigma|$, where $|Q|$ is number of states and $|\Sigma|$ is the alphabet size of the dPFSA.

IV. Space complexity and lower bounds

Minsky construction size is linear in number of states and alphabet size $O(|Q||\Sigma|)$. A more efficient way is to use clever representations other than one-hot – **Indyk** $O(|\Sigma|\sqrt{Q})$ optimal.

Remark: Cannot use Indyk $O(\sqrt{Q})$ for dPFSA. Because the size of an HRNN has a lower bound of $\Omega(|Q|)$ to simulate a dPFSA. The output matrix E is the bottleneck, it has to be large enough to represent all $|Q|$ probability distributions for the transitions leaving state.

V. Turing completeness

Assume arbitrary precision and unbounded computation time. **Theorem:** An Elman RNN with **saturated sigmoid** function is Turing complete. Saturated sigmoid function is $\sigma(x) = 0$ if $x \leq 0$, $\sigma(x) = x$ if $0 < x \leq 1$, $\sigma(x) = 1$ if $x > 1$.

8. Transformer, Tokenization & Generation

Motivation: (1) **Parallelization** - RNNs are inherently sequential, to compute the hidden state h_t , we must first compute h_{t-1} which depends on h_{t-2} . (2) **Info bottleneck** - RNNs must compress entire history of a sequence into a single fixed-size hidden state vector h_t .

I. Transformer sequence model

A language model that uses a Transformer to compute the context encoding h_{t-1} . The probability of the next word is calculates as $p_{SM}(y_t|y_{<t}) = \text{softmax}(Eh_{t-1}) = \text{softmax}(Enc_{\tau}(y_{<t}))_{y_t}$, where τ is a transformer network and $E \in R^{|\Sigma| \times D}$ a symbol representation matrix.

II. The attention mechanism

Let $f: R^D \times R^D \rightarrow R$ be a scoring function and $f_{\Delta D-1}$ a projection function. Furthermore, let

$q \in R^D$, $K_t = (k_1^T, \dots, k_t^T) \in R^{t \times D}$ and $V_t = (v_1^T, \dots, v_t^T) \in R^{t \times D}$. Attention over K_t , V_t also denoted by $Att(q, K_t, V_t): R^D \times R^{t \times D} \times R^{t \times D} \rightarrow R^D$ is a function computing the vector a in the following two-step process: $s_t = (s_1, \dots, s_t) = f_{\Delta D-1}(f(q, k_1), f(q, k_2), \dots, f(q, k_t))$, $a_t = Att(q, K_t, V_t) = s_1 v_1 + s_2 v_2 + \dots + s_t v_t$. q is the query, the current token that is asking; K is the keys, "labels" for the information in the context tokens; V is the values, actual information or representation of these tokens.

Process: (1) **compute scores:** for current q , compute a similarity score against every k_i , common scoring function $f(q, k) = \frac{\langle q, k \rangle}{\sqrt{D}}$ dot product. (2) **compute weights:** normalize these scores using softmax, having attention weights s . (3) **compute output:** sum all value vectors v_i , which is $a_t = Att(q, K_t, V_t) = \sum_{i=1}^t s_i v_i$. **Remark:** Computing the next token conditionals for a string of length N with L layers of soft attention requires $O(NL)$. $O(N^2 L)$ for full sequence. Space complexity is $O(NL D_{model})$.

III. Types of attention

Soft attention: The approach using softmax. **Hard attention:** A theoretical variant where only the value corresponding to the key with the highest score is selected (an argmax operation). (1) $\text{hadmax}_{avg}(x)_d = \frac{1}{|\argmax(x)|}$ if $d \in \argmax(x)$, 0 otherwise. (2) $\text{hardmax}_{uni}(x)_d = 1$ if $d = \hat{d}$, 0 otherwise, $\hat{d} \sim \text{Unif}(\argmax(x))$ or deterministically choose $\hat{d} \in \argmax(x)$. All for $d = 1, \dots, D$ and $x \in R^D$.

Remark: A Transformer LM that uses soft attention is guaranteed to be tight.

IV. Other important concepts

(1) **Positional encodings:** provides the model with information about the position of each token in the sequence (**can be learned**). (2) **Multi-head attention:** runs several attention heads in parallel, each head has own Q, K, V .

V. Simulating n-gram models

Theorem: For any n-gram LM, there exists a Transformer that can simulate it. Transformers can recognize **strictly local languages**.

Construction: (1) **Multi-head attention** with $n - 1$ heads, to find and return the identity of the symbol at position $t - h$. (2) **Positional encoding** such as $f_{pos}(t) = (t, 1)^T$. (3) **Design**

Q, K, V : query from t represents $t - h$ position; key from j represents its own position; scoring function $-| < q_t, k_j > | = -|t - h - j|$ maximized at $j = t - h$, with hard attention forced to attend only to the desired position; value transformation passes through one-hot encoding. (4) $n - 1$ one-hot vectors returned by heads are **concatenated**. A FFD identifies n-gram, outputs probability distribution.

Remark: Powerful as n-gram. More powerful

for context-free and Turing-complete rely on homomorphic equivalence “trick”.

VI. Tokenization & Generation

1. Byte Pair Encoding (BPE) iteratively merges the most frequent adjacent token pairs into new tokens, expanding vocabulary until target size is reached. Tradeoff: (1) Small vocab: Fewer tokenization variations, less model capacity needed, but longer sequences. (2) Large vocab: Shorter sequences, more model capacity & training time needed; can regularize and help in low-resource settings. Multiple tokenization issue: downside is modeling complexity, upside is better handling of misspellings & regularization. Other challenges: Out-of-vocabulary (OOV) words, morphologically rich languages, snicode & script diversity.

2. A sampling adapter maps one probability distribution over a vocabulary to another, adjusting a language model’s output to improve perceived text quality. Examples: TopK – keep top k tokens; TopP – keep top tokens that takes up P% of the probability mass; softmax temperature. Remark: Greedy encoding cannot get global maximum score path. NP-hard, use heuristic like Beam search.

9. Transfer Learning

I. CoVe (Contextualized Word Vectors)

Given a sequence of words w, produces a sequence of context vectors using the pretrained MT-LSTM: CoVe(w) = MT – LSTM(GloVe(w)) = [⋯ C_{t-1}; C_t; C_{t+1} ⋯]. Concatenate as $\tilde{w} = [\text{GloVe}(w); \text{CoVe}(w)]$ as input word embedding to downstream models.

II. ELMo (Embeddings from LMs)

ELMo can generate context-dependent words. Using two stacked, multi-layer bidirectional LSTMs, it has (1) a forward language model reads from left to right $\log p_{LM}(y_t^{(n)} | y_{<t}^{(n)}; \bar{\theta}, \theta')$ and (2) a backward language model reads from right to left $\log p_{LMB}(y_t^{(n)} | y_{>t}^{(n)}; \bar{\theta}, \theta')$. $ELM_o^{\text{task}} = \gamma^{\text{task}} \sum_{l=0}^L s_l^{\text{task}} h_{tl}^{LM}$, context-specific representations $h_{tl}^{LM} = [\bar{h}_{tl}^{LM}; \tilde{h}_{tl}^{LM}]$, $\sum_{l=0}^L s_l^{\text{task}} = 1$ are outputs of softmax on hidden representations, γ^{task} is the scale.

III. BERT (Bidirectional Encoder Representations from Transformers)

Pre-training objectives: (1) Masked Language Modeling (MLM): Randomly mask 15% of tokens and predict them using surrounding context to learn bidirectional representations. (2) Next Sentence Prediction (NSP): Predict if sentence B follows sentence A to capture sentence relationships.

IV. RoBERTa (Robustly Optimized BERT Approach)

Same architecture as BERT, but optimizes the pre-training process: (1) train on larger data,

(2) train on larger batch size, (3) dynamic masking, the pattern changes at each epoch.

V. ALBERT (A Lite BERT)

ALBERT is smaller and faster by using (1) factorized embeddings to reduce parameters, (2) shared weights across Transformer layers, and (3) sentence order prediction instead of NSP for better discourse understanding.

VI. ELECTRA (Efficiently Learning an Encoder that Classifies Token Replacements Accurately)

Improves pre-training efficiency by using (1) a small Generator to replace tokens, (2) a larger Discriminator to detect replaced tokens, and (3) learning from all tokens instead of just masked ones.

VII. Seq2Seq Encoder-Decoder Model (T5)

T5 treats all NLP tasks as text-to-text using a Transformer encoder-decoder. It’s guided by input prefixes and pre-trained with text infilling to predict masked spans, enabling flexible, unified task handling.

10. Parameter Efficient Fine-tuning (PEFT)

As language models grow, full fine-tuning becomes costly and storage-heavy. PEFT tackles this by updating only a small subset of parameters, reducing computation, memory use, and overfitting risks on smaller datasets.

I. BitFit Freeze almost all of the model’s weights and only fine-tune the bias terms in the attention and the feed-forward layers. Achieves strong performance on several benchmarks while being extremely parameter-efficient.

II. Adapters Small, trainable neural network modules inserted into a pre-trained model, usually after the multi-head attention and feed-forward sublayers. Only newly added adapter layers are trained during fine-tuning.

III. LoRA (Low-Rank Adaptation)

Hypothesizes that the change in weights during model adaptation has a “low intrinsic rank.” Approximates the weight update matrix with a low-rank decomposition $\Delta W = BA$, A and B are two small low-rank matrices to update, the pretrained weight matrix W is frozen. Forward pass is $h = Wx + \frac{\alpha}{r} BAx$. Merge update to original weights $W' = W + BA$ without extra inference latency. Efficient and flexible.

11. Prompting

Name	Notation	Example	Description
Input	x	I love this movie.	One or multiple texts
Output	y	++ (very positive)	Output label or text
Prompting Function	$f_{\text{prompt}}(x)$	[X] Overall, it was a [Z] movie.	A function that converts the input into a specific form by inserting the input x and adding a slot [Z] where answer x may be filled later
Prompt	x'	I love this movie. Overall, it was a [Z] movie.	A text where [Z] is instantiated by input x at answer slot [Z] is not.
Filled Prompt	$f_{\text{fill}}(x', a)$	I love this movie. Overall, it was a good movie.	A prompt where slot [Z] is filled with any answer.
Answered Prompt	$f_{\text{fill}}(x', a')$	I love this movie. Overall, it was a good movie.	A prompt where slot [Z] is filled with a true answer.
Answer	a	“good”, “fantastic”, “horror”	A token, phrase, or sentence that fills [Z]

I. In-context learning Lets large language models perform new tasks using only the prompt, without updating weights. Zero-shot uses just task instructions, while few-shot adds examples to guide the model, performs better as model size increases.

II. Prompting Strategies

1. Chain of Thought (CoT): Step-by-step. 2. Least-to-Most: Break to subproblems. 3. Self-Consistency: Samples multiple reasoning paths and selects the most consistent final answer (e.g., majority vote) for more accurate results.

12. Multimodal Language Models

Inspired by LLM success in NLP, Vision-Language Pre-training (VLP) is the standard of linking visual and textual understanding.

I. VLM Architectures (key components)

A typical VLM includes a text encoder, a vision encoder, a fusion module, and sometimes a decoder. (1) Vision Encoder: Extracts image features via Object Detectors (e.g., Faster R-CNN), CNNs (e.g., ResNet), or Vision Transformers (ViT). (2) Text Encoder: Uses pre-trained models like BERT to convert text into feature vectors. (3) Fusion Module: Combines visual and text features using merged attention or co-attention.

II. Image-Text Contrastive Learning (ITC)

Used by CLIP, ITC trains the model to align image & text vectors by identifying the correct pairs within a batch. The goal is to maximize similarity between matching image-text pairs while minimizing it for all others.

III. CLIP (Contrastive Language-Image Pre-training)

Training: Contrastive pre-training to align image and text embeddings. Testing: Retrieve captions by highest cosine similarity to the image. (Optional) Generate synthetic captions from object classes. Objectives: Predict matched image from caption and vice versa. Maximize cosine similarity for matched pairs, minimize for mismatched pairs in the batch.

13. Instruction Tuning & RLHF

I. Instruction Tuning Fine-tunes pre-trained language models on diverse NLP tasks using natural language instructions.

A task description is prepended to inputs during training (e.g., "Is the sentiment positive or negative? Review: [text]"). Training on many instructed tasks helps the model follow instructions and generalize to unseen tasks.

II. RLHF

Supervised Policy (SFT): Fine-tune on human demos. Reward Model: Train on human-ranked best-of-N outputs using Bradley-Terry loss $L(\theta_{RM}) = \frac{1}{(\frac{1}{2})^2} E \left[\log \left(\sigma \left(r_{\theta_{RM}}(x, y_u) - r_{\theta_{RM}}(x, y_l) \right) \right) \right]$ to score responses. PPO: Use RM rewards to

fine-tune policy with KL penalty to avoid drift. DPO: Directly fine-tunes on preference pairs without a RM or RL, optimizing likelihood of preferred over dispreferred relative to SFT.

14. Retrieval-Augmented Generation (RAG)

I. Two kinds of RAG models Parametric:

Store knowledge in model parameters, which is static and costly to update. Non-parametric: Use an external, updatable knowledge base to retrieve relevant information at inference.

II. Benefits of RAG Keeps knowledge current. Reduces hallucination by grounding outputs in real documents. Improves interpretability. Broader, long-tail knowledge. More efficient than large parametric models. III. Retrieval Methods Sparse retrieval: TF-IDF and BM25 rely on exact keyword but struggle with synonyms. Dense retrieval: Embeds queries and documents into a shared vector space for better semantic matching via dot product.

IV. Using Retrieved Information kNN-LM

retrieves similar contexts to adjust output probabilities with dynamic gating. Prompting appends retrieved docs to inputs for generation. Joint fine-tuning (REALM, RETRO) trains retriever and generator together, fusing info for better results.

15. Attacks

I. Adversarial attacks and defenses

Definition: Adversarial examples are slight input tweaks designed to fool models. Gradient-based (white-box) / black-box (no gradient) transfer attacks, query-based attacks. 1. Jailbreaking: Craft inputs that coax the model into harmful responses by starting with agreeable phrases.

2. GCG (greedy coordinate gradient) Attack: Adds gibberish adversarial suffixes optimized via gradients to increase jailbreak chances, transferable across models.

3. Solutions: Perplexity Filters: Reject unnatural, high-perplexity prompts. Jailbreak Detectors: Use models to flag unsafe prompts or outputs. Adversarial Training: Train on adversarial examples for robustness.

Representation Engineering: Monitor internal activations to prevent unsafe outputs.

II. Prompt Injections Attackers embed instructions within data that override original prompts because LLMs can’t distinguish instructions from data. Injected text overrides translation instructions. Malicious calendar events trick assistants into leaking data. Delimiters: Mark data separately, though not foolproof. Instruction Hierarchy: Prioritize system prompts over user data. System Isolation: Use two LLMs one reads trusted data; the other reads untrusted data.

III. Data Poisoning & Backdoors Attackers add poisoned data with triggers to training sets to create harmful backdoors, causes the model to produce specific, unwanted outputs. Can poison RLHF by mislabeling harmful outputs as good, or poison pre-training data stealthily. Add SHA hash codes, more checks.

16. Watermarking & Model Stealing

Watermarking embeds a secret, statistically detectable signal into AI-generated text to prove its origin.

I. (Hard) red list watermarking

Randomly partition the vocabulary into “green” and “red” lists for each generated token. Logits ($l_k^{(t)}$) converted to probability distribution using $p_k^{(t)} = \frac{\exp(l_k^{(t)})}{\sum_i \exp(l_i^{(t)})}$. The model is restricted to select tokens from the green list. The probability of a natural writer producing T tokens is 0.5^T . Use z-test to evaluate if the text is watermarked $z = \frac{2(|s|_G - \frac{T}{2})}{\sqrt{T}}$, $|s|_G$ is the number of green list tokens used. We can model as $X \sim \text{Binomial}(T, 0.5)$, expectation of green tokens is $E[X] = np = \frac{T}{2}$, variance $\text{Var}(X) = np(1 - p) = \frac{T}{4}$.

II. Soft red list watermarking

$$p_k^{(t)} = \begin{cases} \frac{\exp(l_k^{(t)} + \delta)}{\sum_{i \in R} \exp(l_i^{(t)}) + \sum_{i \in G} \exp(l_i^{(t)} + \delta)}, & k \in G \\ \frac{\exp(l_k^{(t)})}{\sum_{i \in R} \exp(l_i^{(t)}) + \sum_{i \in G} \exp(l_i^{(t)} + \delta)}, & k \in R \end{cases} \text{ by adding}$$

constant δ to logits of green tokens, the expected number of green tokens now depends on the entropy of tokens in the sequence.

III. Model stealing

Cloning a target model using only black-box access. Logistic regression, solve linear system with $d + 1$ queries to get w, b . Query along line to find $d + 1$ decision boundary. Transformers, hidden dim h : rank of logit matrix Q from random prompts; embedding W , SVD on Q recovers W ; use logit bias trick, subtract bias to get full logits from top-k.

17. Privacy to Satisfy by an LLM

Private learning defends against data extraction and membership inference attacks that exploit model memorization. Heuristic defenses like deduplication and filters can fail or leak info. Differential Privacy (DP) offers a formal guarantee by adding noise (e.g., via DP-SGD) to ensure outputs reveal little about any single data point. Public pre-training with private fine-tuning, federated learning, and secure multiparty training improve privacy but remain vulnerable to gradient attacks. Encoded-data methods and privacy backdoors have flaws. Membership inference uses loss patterns to detect training data membership; random canaries and extracting unique strings help prove data inclusion.

