MS&E 319

Efficient Generative Language Models

Amin Saberi

Professor, Management Science and Engineering and Computer Science (by courtesy)
Director of Stanford Lab on Language, Data, and Reasoning
Stanford University

My Co-instructors

The course combines theoretical, algorithmic, and systems perspectives on generative language models.

Anay Mehrotra

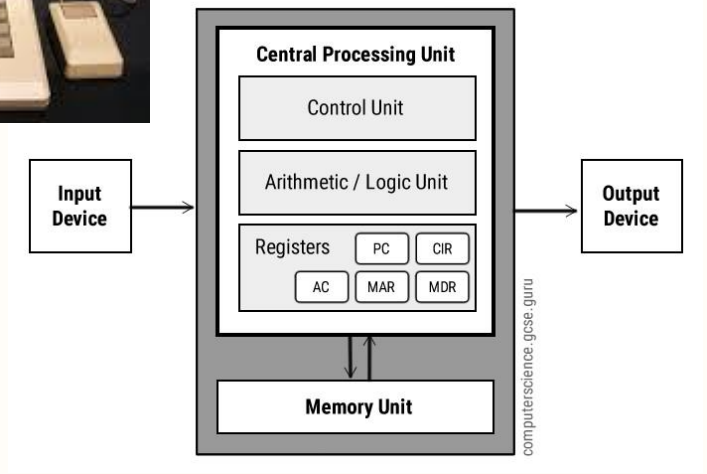
Motwani Postdoctoral Fellow
Stanford University

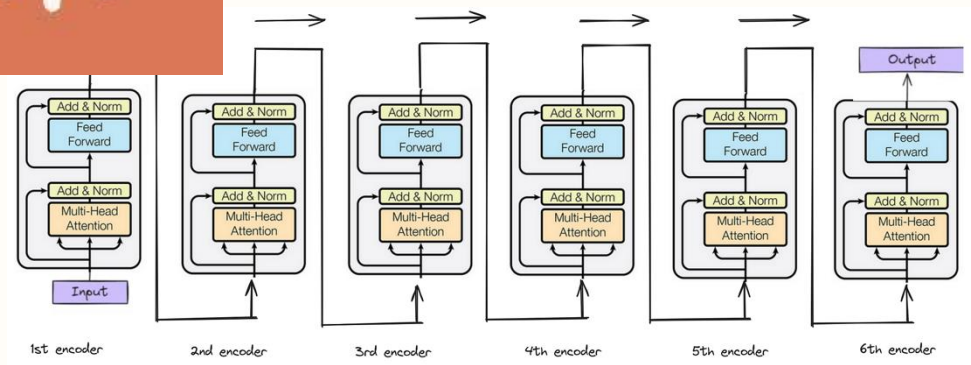
Amin Karbasi

VP and Chief AI Scientist, Cisco
Adjunct Professor, Stanford

Grigoris Velegkas

Research Scientist
Google Research


Central Processing Unit
Control Unit
Arithmetic / Logic Unit
Registers
PC
CIR
AC
MAR
MDR
Memory Unit
Input Device
Output Device
computerscience.gcse.guru


Add & Norm
Feed Forward
Multi-Head Attention
Input
Output
1st encoder
2nd encoder
3rd encoder
4th encoder
5th encoder
6th encoder

Designing generative language models is an empirical science

Much like the natural sciences, such as physics and chemistry, it needs both experimental and theoretical components.

EXPERIMENTS

Measure quality, compute, memory, and latency.

Reveal sensitivity to data, scale, and implementation.

Establish performance in the tested regime.

Theoretical analysis

Derive consequences from stated assumptions.

Identify the structure an algorithm exploits.

Predict how behavior should change outside the observed regime.

Experiments establish what works in the regime tested. Theory narrows the search and predicts what may transfer to larger size/different parameter settings.

Learning objectives for this course

Students should be able to explain a method's foundations, identify its algorithmic saving and tradeoffs, and formulate a falsifiable extension.

01	**FOUNDATIONS**	Explain the mechanism and its assumptions. State whether the argument is an exact derivation, an approximation, or an empirical regularity.
02	**ALGORITHM**	Explain how the algorithm reduces computation or memory. Identify the approximation, assumption, or loss in quality that buys the saving.
03	**RESEARCH**	Change an assumption, predict the consequence, and design an experiment that could falsify the prediction.

Course outline

Part I: Innovations in pre-training

Pre-training asks how to turn a fixed budget of data and computation into useful, transferable capability.

Examples of topics

- **Architecture and objective.** Review the language model training pipeline
- **Scaling laws.** Use measured loss curves to divide a compute budget between model size and training data, and learn when experiments at smaller scale predict behavior at larger scale.
- **Training by predicting several future tokens.** Ask whether supervising several next positions produces a better representation, and whether the gain justifies the additional computation.
- **Matrix-aware optimization.** Study optimizers such as Muon (& others) that treat a weight matrix as a linear map, then compare methods by the time and computation needed to reach a fixed quality.

Example: Scaling Laws

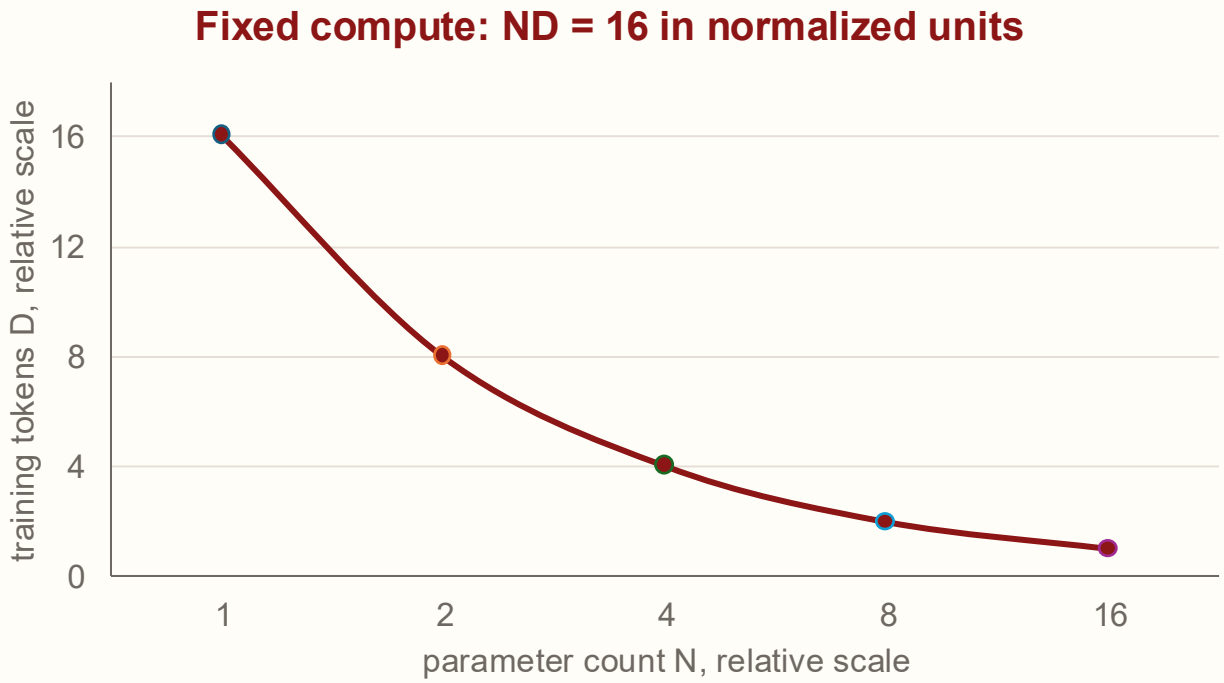
Scaling laws measure how performance changes with model size and training data, then turn those measurements into an allocation rule for a fixed compute budget.

Training compute: an allocation problem

Fix a pretraining budget: increasing parameter count forces model to see fewer training tokens.

$$C_{\text{train}} \approx \kappa ND$$

- C_{train}: training compute
- N: parameter count
- D: training tokens



Kaplan et al. (2020); Hoffmann et al. (2022)

Scaling laws produce an allocation rule

An empirical loss model estimates the returns to parameters and data. Blue variables are estimated using data.

$$L(N,D) = L_\infty + AN^{-\alpha} + BD^{-\beta}$$

- L_∞: irreducible loss
- A: parameter count
- N: parameter count
- B: training tokens
- D: training tokens

$$C = \kappa ND$$

- C: compute budget

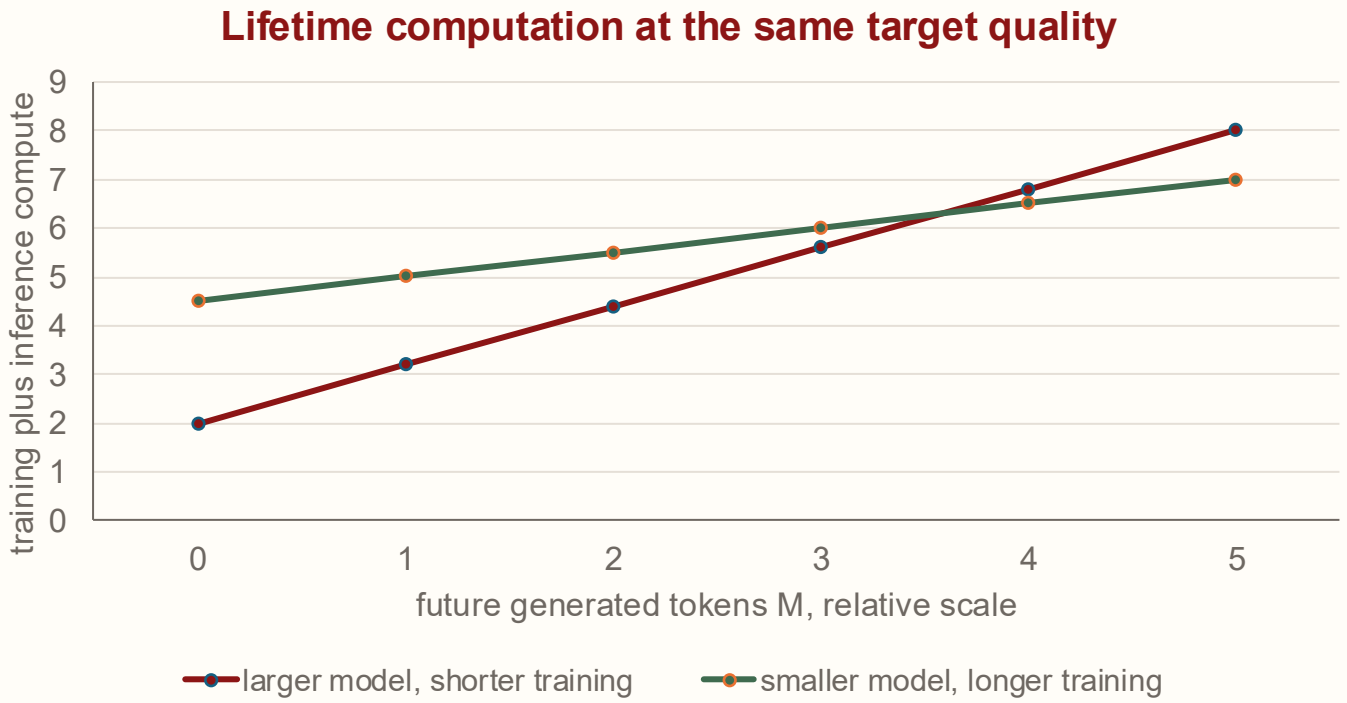
THE OPTIMIZATION

Substitute D = C/(κN). The optimum balances the marginal benefit of adding parameters against the marginal benefit of adding data.

Kaplan et al. (2020); Hoffmann et al. (2022)

Deployment demand matters too!

Extra training can be worthwhile when it produces a smaller model with lower cost for every future token.



$$J = C_{train} + M\, c(N)$$

- lifetime compute
- training compute
- future tokens
- inference per token at model size N

The smaller model pays more during training and accumulates inference cost more slowly.

Sardana et al. (2024), Beyond Chinchilla-Optimal

Part II: Innovations in model architecture

We will consider architecture choices that influence which tokens activate, which context is accessible, and what state the system should retain.

Examples of topics

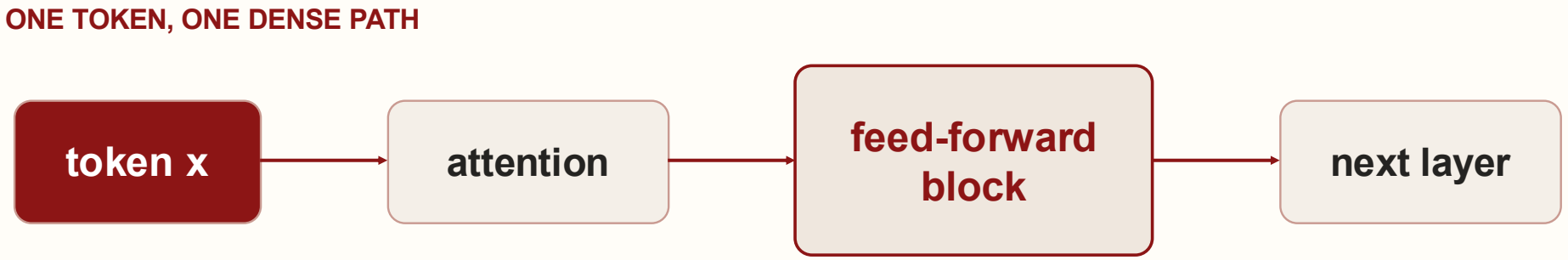
- **Mixtures of experts.** A router sends each token to only a small subset of stored experts. We will study specialization, shared experts, load balancing, and communication cost.
- **Attention design.** Sparse attention restricts which positions interact. Linear attention summarizes the past. FlashAttention computes dense attention while reducing memory traffic. These methods change different parts of the system.
- **Memory for previous tokens.** Compare multi-query attention, grouped-query attention, and multi-head latent attention as different choices about which keys and values to retain for future tokens.

Example: Mixture of Experts

Dense models use the full feed-forward network for every token. A mixture of experts stores many feed-forward networks but routes each token through only a few, separating stored capacity from active computation.

Dense transformers tie capacity to computation

In a dense transformer, every token uses every parameter in each feed-forward block.



$$F(x) = W_2\,\varphi(W_1 x)$$

layer output

learned matrices

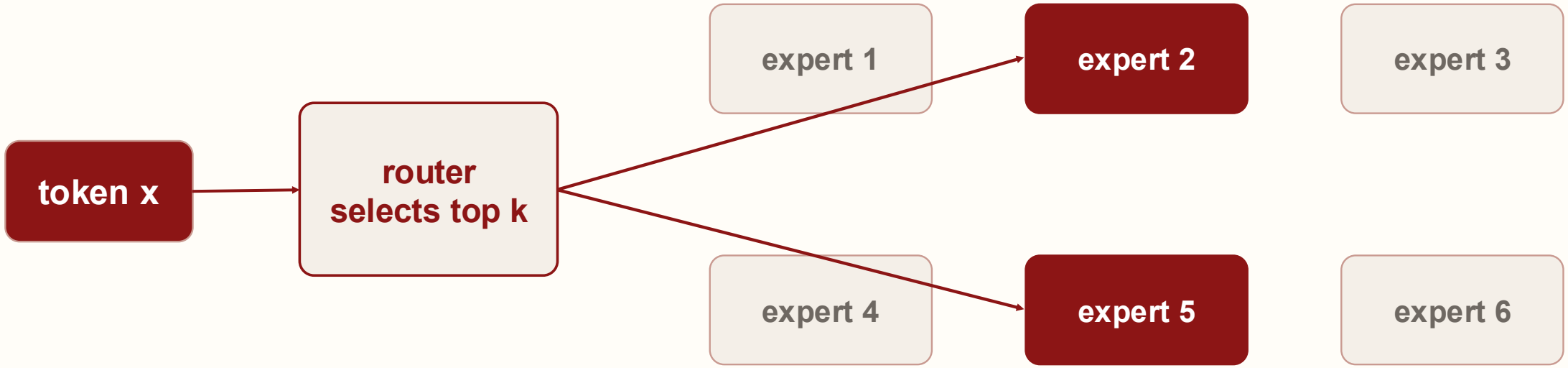
Stored parameters & computation per token:
proportional to hidden dimension x width

Increasing feed-forward width increases what layer can store and what every token must compute.

Vaswani et al. (2017); Shazeer et al. (2017)

Mixture of experts decouples capacity and compute

A mixture-of-experts layer stores several feed-forward networks but sends each token to only a few

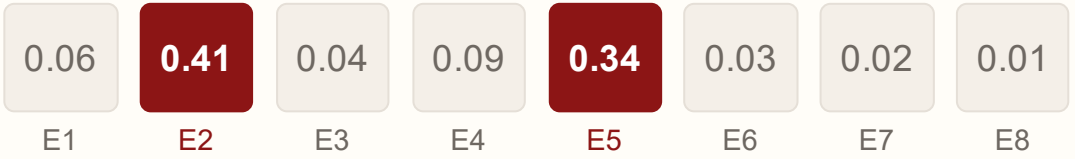


Shazeer et al. (2017); Fedus, Zoph & Shazeer (2022)

The router chooses the right experts for each token

The router scores every expert, keeps the k largest scores.

ROUTER SCORES FOR ONE TOKEN



$$T(x) = \mathrm{TopK}(W_r x, k)$$

selected expert set · **token state** · **router matrix** · **experts chosen**

TopK tries to choose the best k experts for the token

A CONCRETE SYSTEM

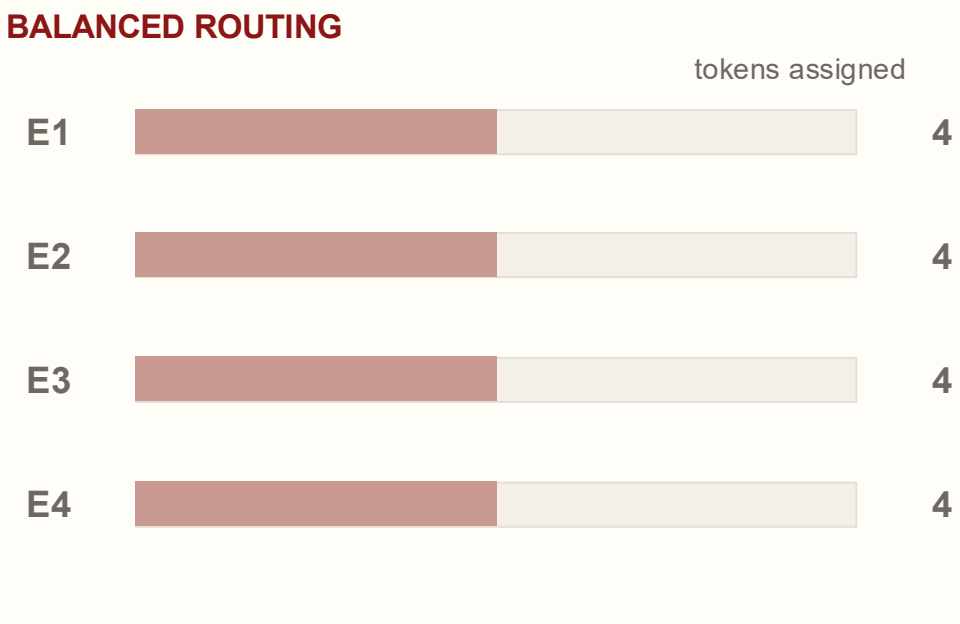
Mixtral of Experts

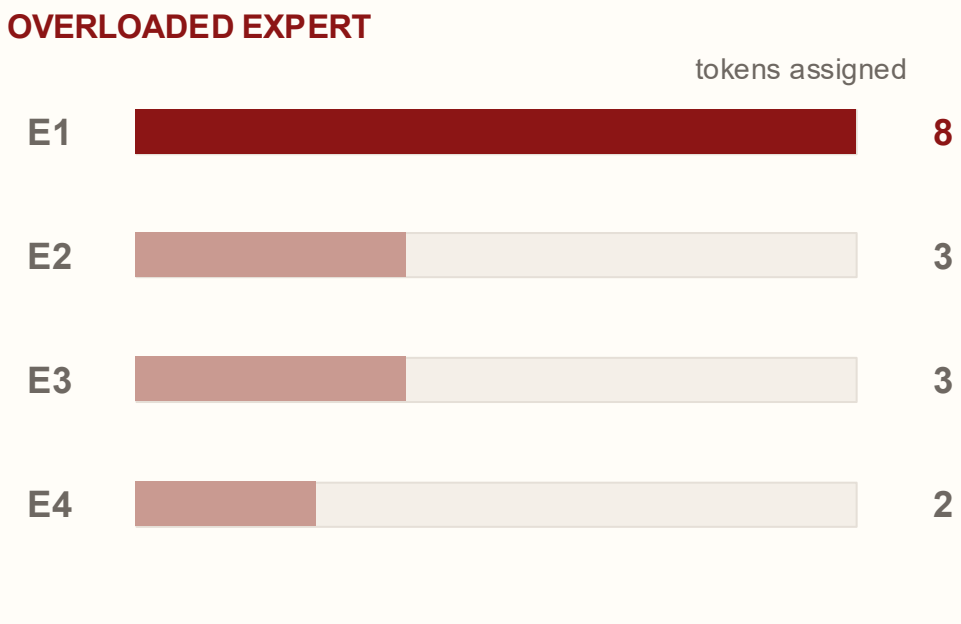
8 feed-forward experts in each sparse layer
2 experts selected for each token
47 billion total parameters
13 billion active parameters for one token

Fedus, Zoph & Shazeer (2022); Jiang et al. (2024)

Sparse experts: algorithmic questions

Load balance, memory, and communication determine inference (wall-clock) time.





Fedus, Zoph & Shazeer (2022); Lepikhin et al. (2021)

Part III: Innovations in efficient inference

Efficient inference asks how to reduce memory, improve speed, and lower cost while preserving the quality and behavior that matter for deployment.

Examples of topics

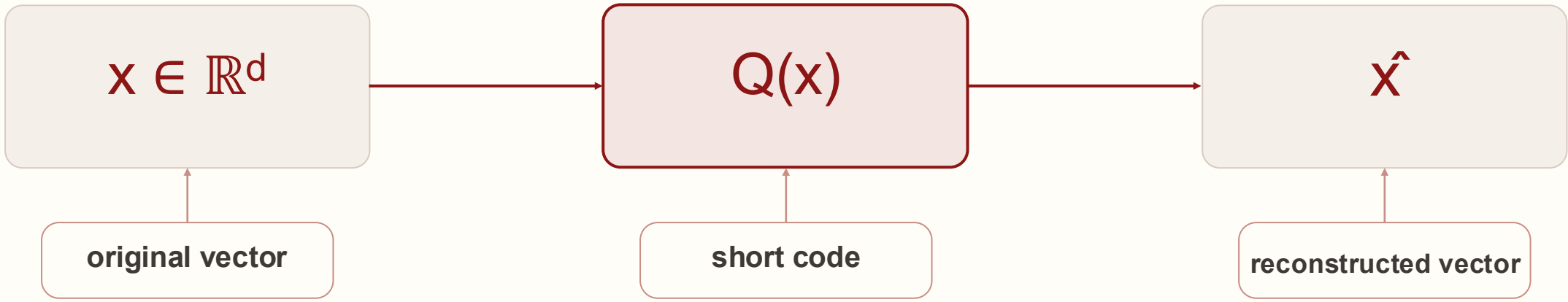
- **Quantization and TurboQuant.** Store weights, activations, and cached state with fewer bits. Study how rotations and residual corrections reduce the error that matters for inner products and model outputs.
- **Speculative decoding.** A smaller draft model proposes several tokens. The target model verifies them in parallel and corrects rejected proposals without changing the target distribution.
- **Learned cache compression.** Train a compact representation of a reusable corpus, then measure how often reuse repays the preparation cost and whether rare but important information survives.

Example: Quantization

Quantization reduces the cost of inference. Inference repeatedly moves large arrays of numbers. Shorter numerical representations reduce both storage and memory traffic.

Quantization replaces a vector with a short code

A quantizer maps a real vector to a finite code, then reconstructs an approximation when the vector is needed.



The design question is which information the short code must preserve.

A good code preserves crucial geometric properties

The coordinates may change after compression, but the distances and similarities used by the model should change little.

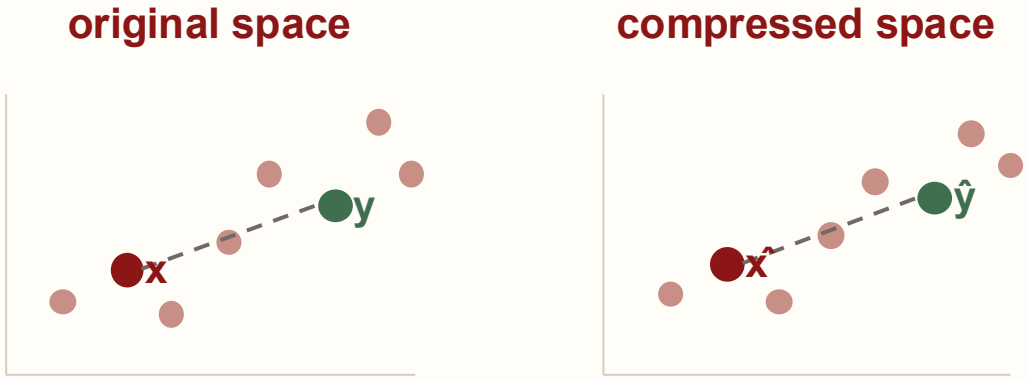
LOW DISTORTION EMBEDDING

A compact representation that approximately preserves geometric relationships.

$$\|x - y\|_2 \approx \|\hat{x} - \hat{y}\|_2$$

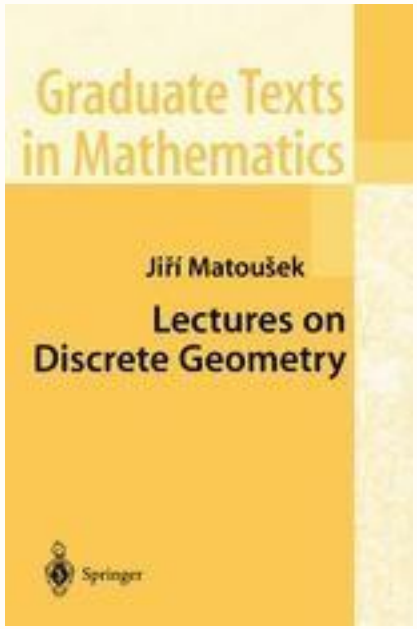
$$\langle x, y \rangle \approx \langle \hat{x}, \hat{y} \rangle$$

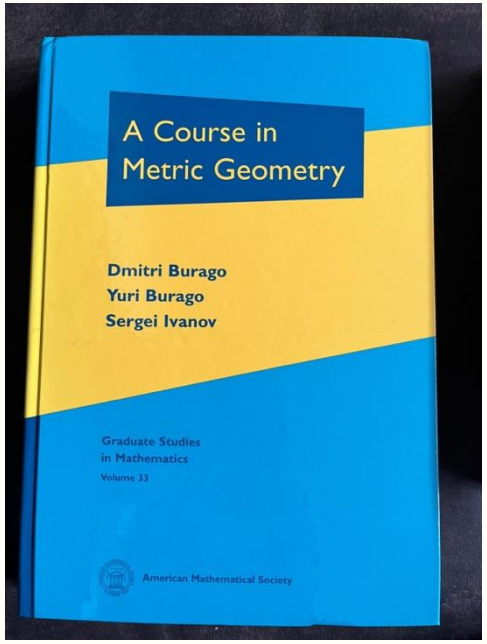
x, y: original vectors $\hat{x}$, $\hat{y}$: reconstructions



nearest neighbors and attention scores are preserved

Metric embedding is studied extensively in mathematics





One more idea: Rotation makes quantization easier

A few unusually large coordinates can dominate rounding error. A random rotation spreads the vector's energy more evenly.

$$z = R\,x$$

$$\|z\|_2 = \|x\|_2$$

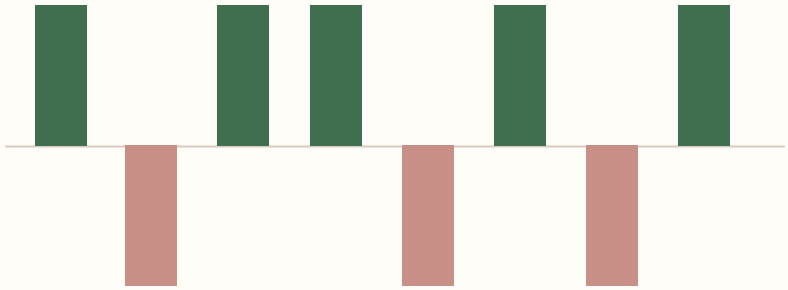
- **x** original vector
- **R** random rotation
- **z** rotated vector

Rotation changes the coordinates while preserving lengths, angles, and inner products.

before rotation

energy concentrated

after rotation

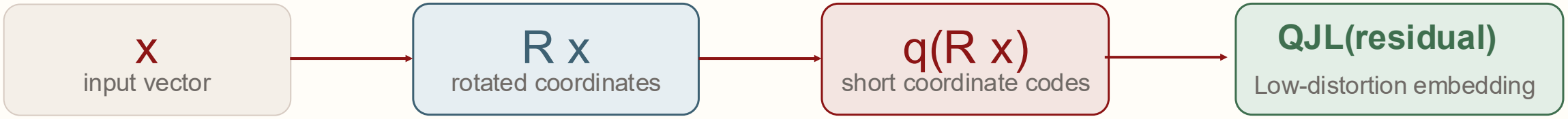


energy spread across coordinates

The rotation preserves geometry exactly but makes coordinatewise rounding less fragile.

TurboQuant: putting it all together

TurboQuant first rotates each vector, then applies an optimized scalar quantizer to every coordinate.



Theoretically: achieved error decay within a small constant of the information theoretic optimum.

In experiments: matched full precision long-context performance using only 3.5 bits per channel and achieved near perfect retrieval with more than (4x) cache compression.

Zandieh et al. (2025)

Part IV: Innovations in post-training

Post-training turns a pretrained model into a useful assistant and adapt it to particular tasks and values.

Examples of topics

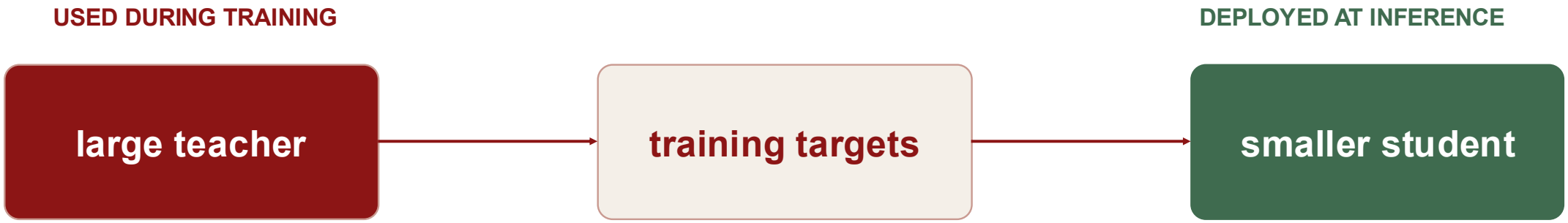
- **Supervised fine-tuning.** Train on curated prompt-response examples to teach formats, tasks, and desired behavior. Low-rank adaptation restricts changes to a small trainable update while keeping the base model fixed.
- **Learning from preferences.** Use comparisons between responses either to fit a reward model and optimize a policy, or to train the policy directly from the comparisons.
- **Distillation and self-distillation.** Train on probabilities or selected solutions from a stronger model, or from the learner's own filtered generations. Improvement depends on how the targets are produced and selected.
- **Recursive improvement.** Ask when models can help generate, judge, and refine future training data, and when evaluator errors or narrow supervision cause the process to stall.

Example: Distillation

Distillation trains a smaller student to imitate a larger teacher, preserving much of its capability at a lower inference cost.

Distillation transfers behavior to a smaller model

A large teacher produces training targets. A smaller student learns to imitate its answers and serves future queries at lower cost.

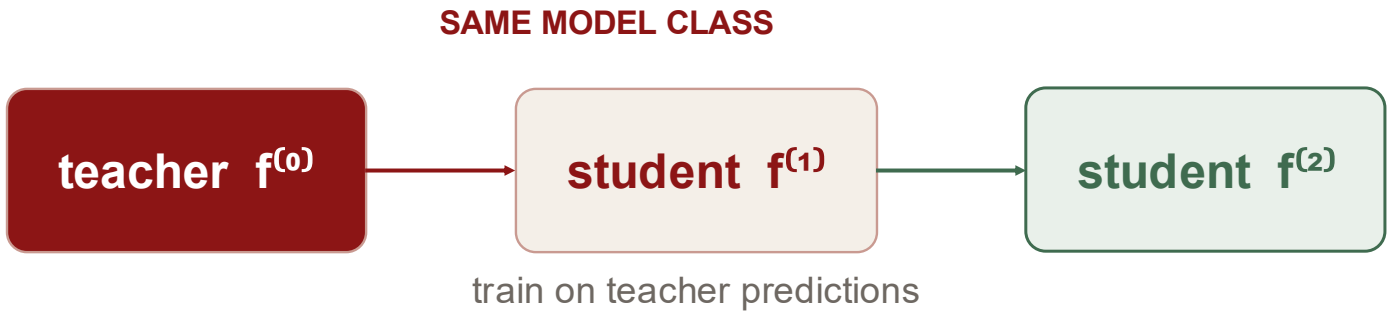


Spend more computation during training to use less computation at inference.

Hinton, Vinyals & Dean (2015), Distilling the Knowledge in a Neural Network

Can a student outperform its teacher?

Paradoxically, yes! With the same architecture and no new labels, a self-distilled student can generalize better than its teacher.



A SIMPLE SPECTRAL MODEL

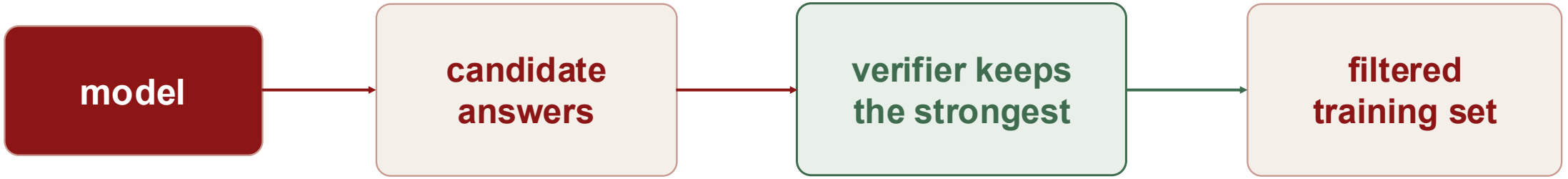
$$\hat{y}^{(t)} = S^t y$$

- predictions after t rounds
- learning operator
- original targets

Furlanello et al. (2018); Mobahi, Farajtabar & Bartlett (2020); Nagarajan et al. (2023)

Self-distillation improves through selection

The model generates candidate answers. A verifier keeps the strongest answers, and the model learns from the filtered set.



Selection turns generated answers into useful supervision.

Zelikman et al. (2022), STaR: Bootstrapping Reasoning With Reasoning

Part V: Risks and vulnerabilities

The final part of the course studies how language models fail under attack, what information their interfaces can reveal, and what guarantees defensive mechanisms can actually support.

Examples of topics

- **Jailbreaking.** Compare attacks that use gradients, black-box search, or model-generated strategies. Judge results only after specifying the attacker's access, query budget, and success criterion.
- **Model extraction.** Determine which model components or properties can be inferred through an application programming interface, and distinguish recovery of one component from reconstruction of an entire model.
- **Text watermarking.** Bias token generation using a secret rule, then test whether the resulting statistical signal survives short text, editing, paraphrase, and spoofing.

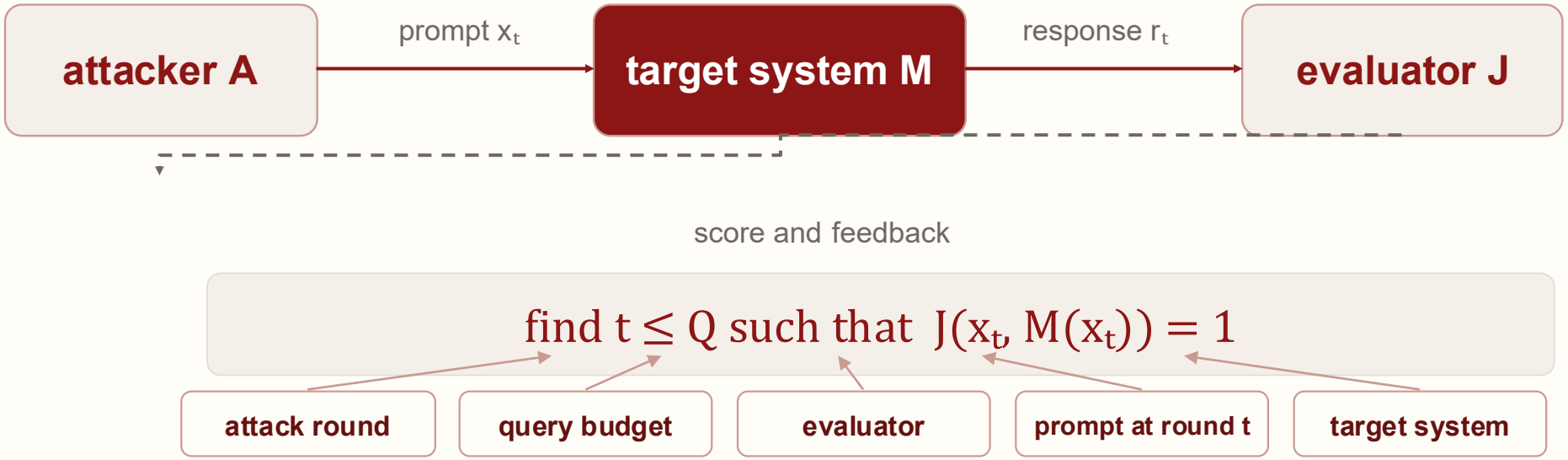
Example: Jailbreaking as Adaptive Search

A jailbreak searches for an input that makes a language model violate its intended behavioral restrictions.

Safety claims depend on the attacker

A refusal to one direct request measures one prompt, not robustness against an adaptive search process.

A jailbreak is a prompt that elicits behavior the deployed system was designed to refuse.



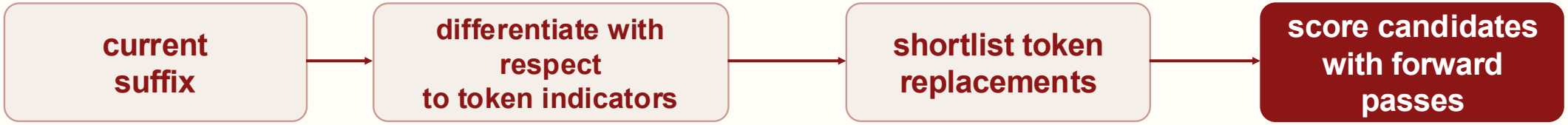
Mazeika et al. (2024), HarmBench

Jailbreaking as a discrete search

Greedy Coordinate Gradient uses derivatives to identify promising token changes, then evaluates the discrete candidates with forward passes.

$$s^* \in \arg\min_s L(s), \qquad L(s) = -\log p_\theta(\tau \mid q \,\|\, s)$$

- loss L(s)
- target model pθ
- target prefix τ
- restricted request q
- candidate suffix s



White-box access supplies gradients. Transfer to another model remains an empirical result, not a guarantee.

Zou et al. (2023), Universal and Transferable Adversarial Attacks

A language model can search the prompt space

Prompt Automatic Iterative Refinement follows one feedback-driven chain; Tree of Attacks with Pruning explores several branches and discards weak ones.

PROMPT AUTOMATIC ITERATIVE REFINEMENT

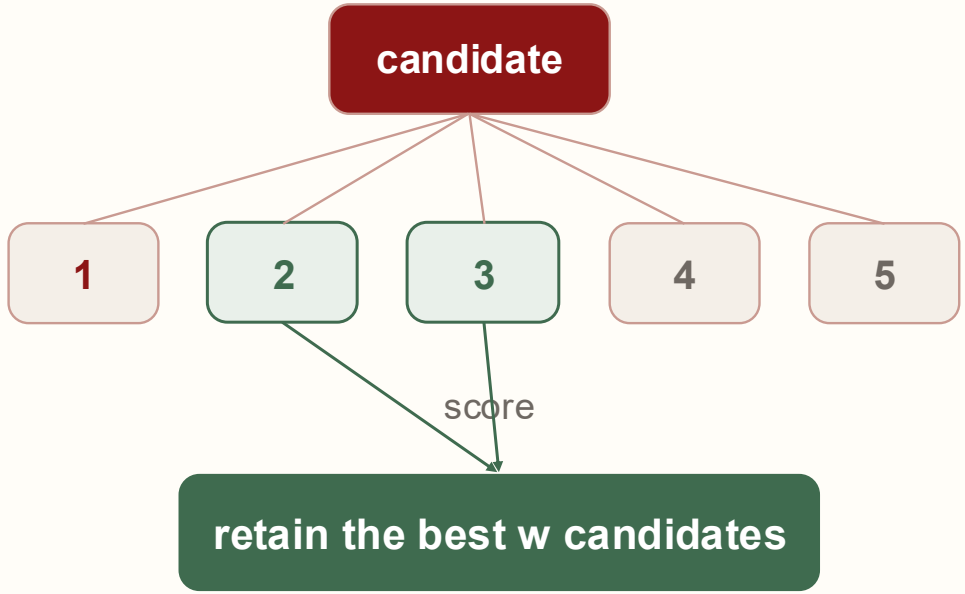
one adaptive chain

target response + evaluator score refine the next candidate

Branching factor b = 1

TREE OF ATTACKS WITH PRUNING

branch, score, prune, continue



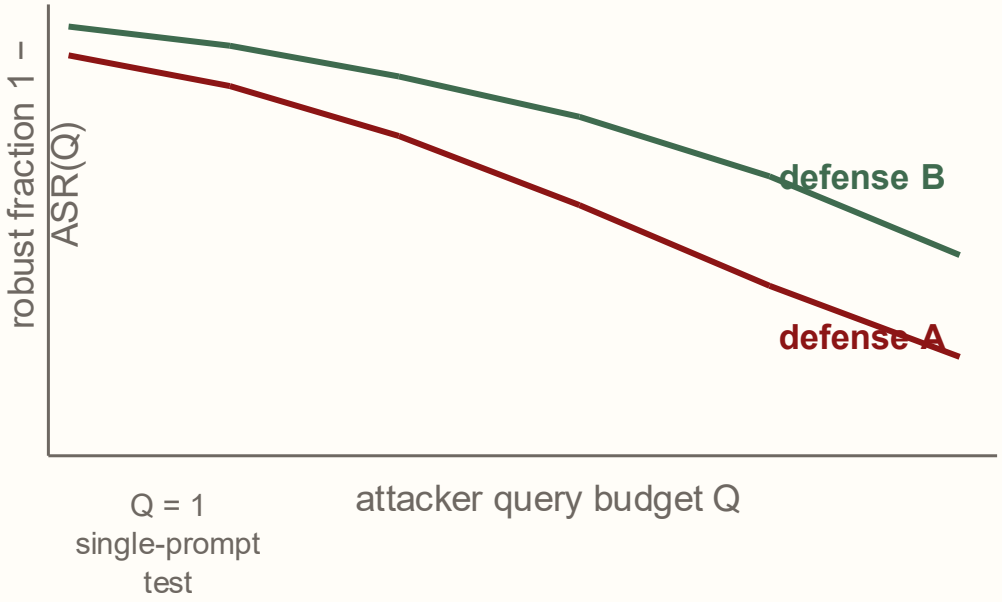
Chao et al. (2023); Mehrotra et al. (2024)

Robustness is a curve over attacker capability

A single attack success rate conceals the access, budget, behavior set, and evaluator that produced it.

$$\text{attack success rate } \mathrm{ASR}(Q) = (1/n) \sum_i \mathbb{1}\{\exists\, t \le Q : J_i(x_{it}, M(x_{it})) = 1\}$$

SCHEMATIC ROBUSTNESS CURVES

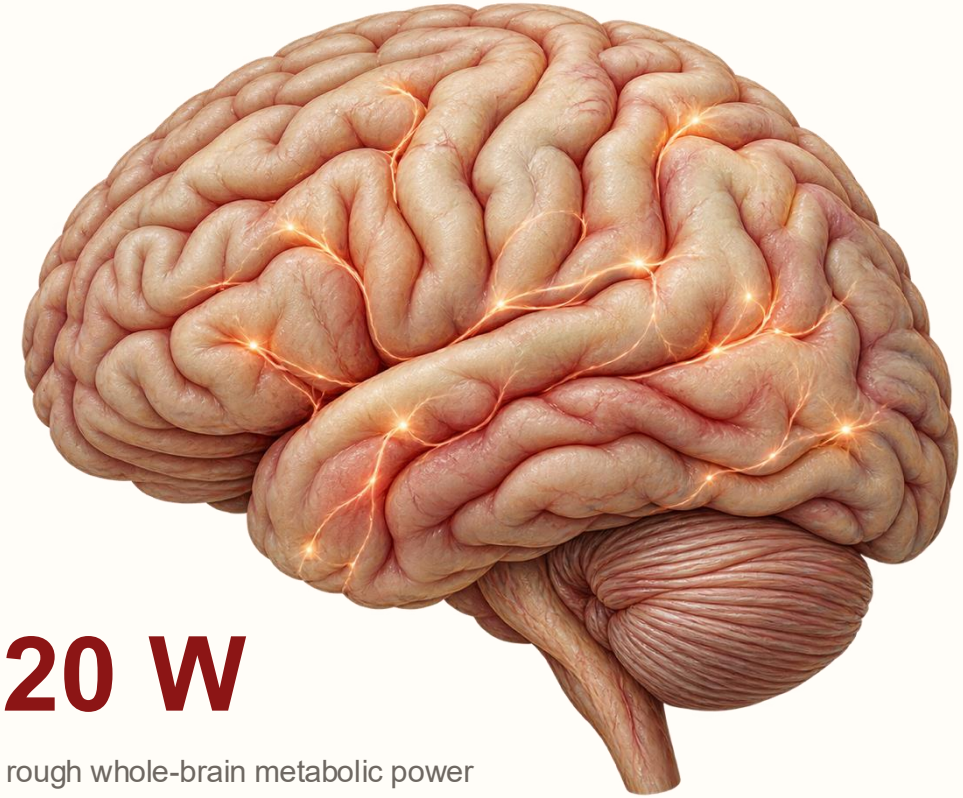


EVERY RESULT DEPENDS ON

behavior distribution, attacker access, query budget, decoding rule and safety wrapper, evaluator and success threshold

Mazeika et al. (2024); Chao et al. (2023)

Twenty watts



20 W

rough whole-brain metabolic power

Intelligence may admit far more efficient implementations than today's systems.

The goal of this course is to identify the ideas that make generative language models more efficient and understand when those ideas survive at scale.

Next:

Review the syllabus
Hear from Anay, Amin K, and Grigoris

The end