IMT SCUOLA ALTI STUDI LUCCA

Languages, Machines & Models

Cosimo Perini Brogi
IMT School for Advanced Studies Lucca

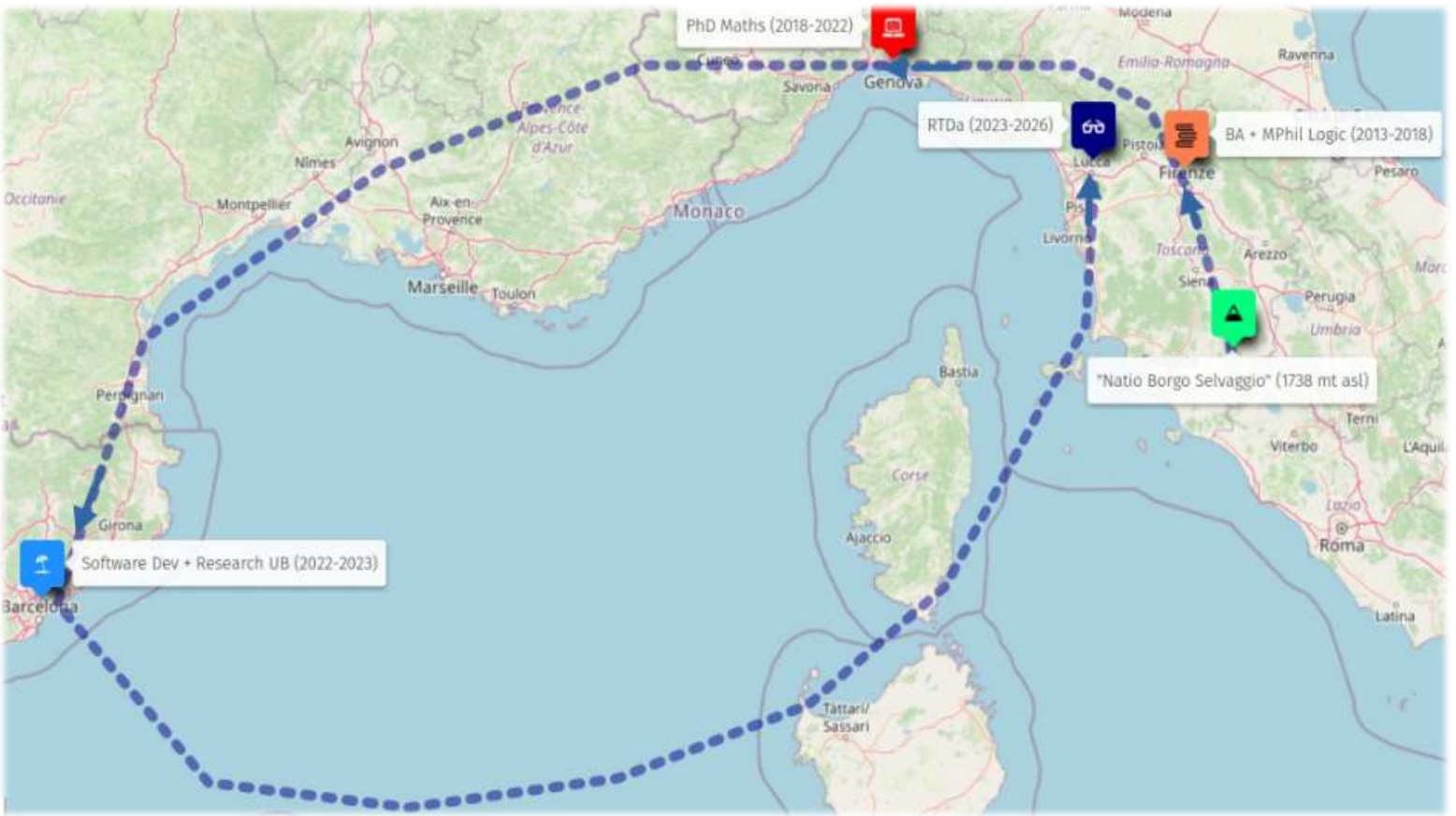
Corso di Perfezionamento in *Intelligenza Artificiale e Discipline Umanistiche*
Università degli Studi di Firenze
16 Ottobre 2025

Overview

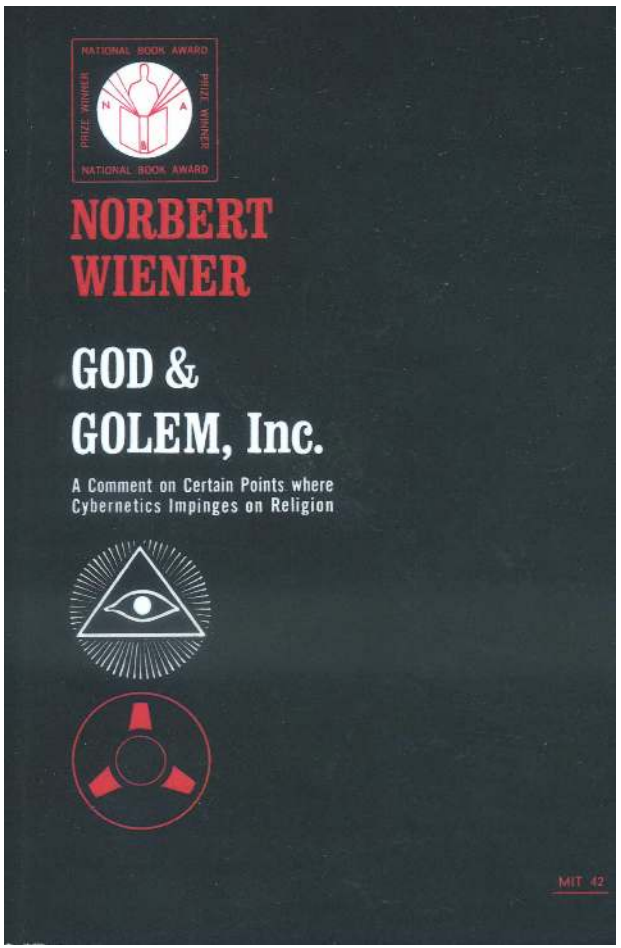
1) **Introduction**
   - Why Me? Why Not
   - This Talk
2) **The Origins**
   - Symbolic Artificial Intelligence
   - Turing's Computing Machinery
   - Linguistic AI in the '60s
   - An Euphoric Analogy
3) **From Units to Transformers**
   - Neural Network Foundations
   - Convolutions and Vision
   - Monte Carlo Tree Search
   - Generative Adversarial Networks
   - Transformers and Large Language Models
4) **How Deep the Rabbit Hole Goes**
   - Limits of Statistical Predictions
   - Harris' Distributions and Chomsky's Structures
   - Language: Biology/Culture
   - Need for Maths
5) **Conclusions**
   - Take Home Message
   - Back to the Analogy

Minimal Bio

- Assistant Professor of Computer Science at IMT Lucca



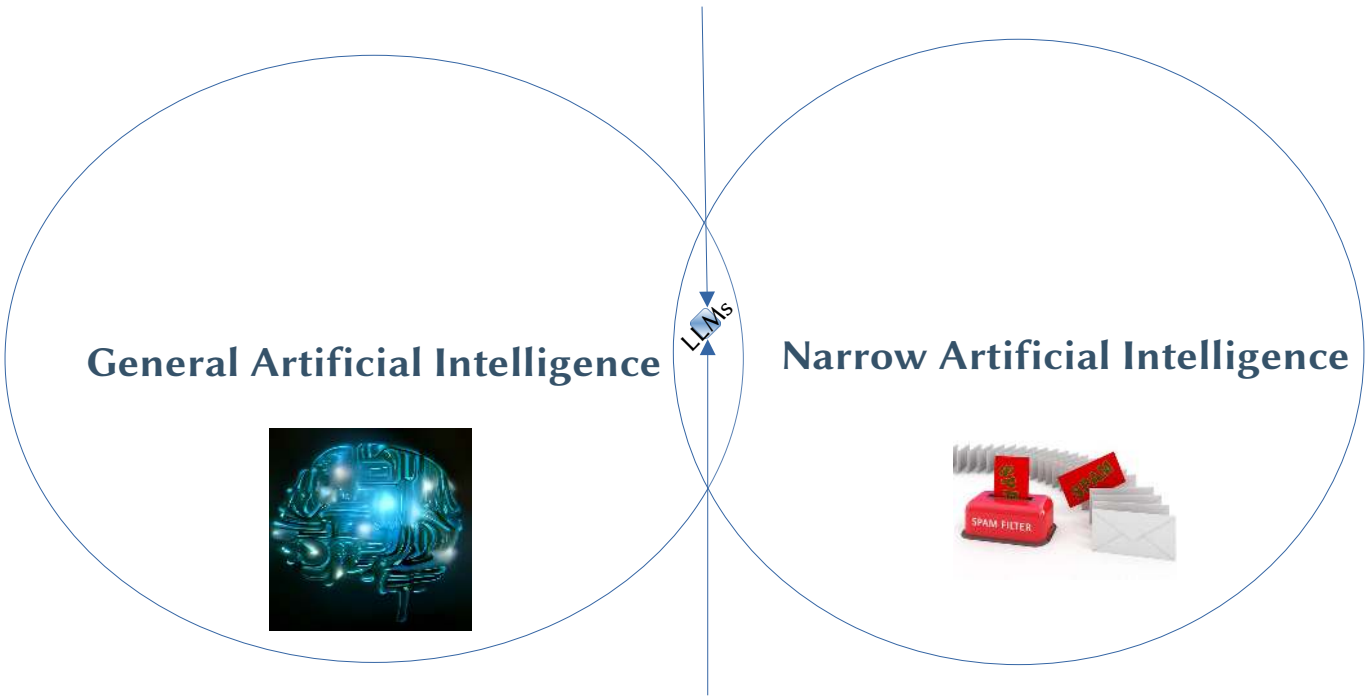
Wiener's Warning



The Origins

We are Here...

Large Language Models



Machine language manipulation and generation of sentences in different (human) languages
- BERT-style: complete missing words from a sentence
- GPT-style: Add next word

… But We Started from There …

The journey into AI began in the **1950s** with a quest of *pure logic*: create an "automatic reasoner", a machine that could follow the rules of formal logic to solve complex problems, *including linguistic interactions*.

The **Logic Theorist** (1956) was designed to automatically prove theorems from the landmark text on mathematical logic, the *Principia Mathematica*.

It used *heuristics* (rules of thumb) to successfully prove 38 of the first 52 theorems it was tasked with, famously discovering *a more elegant proof* for one of them than the one created by its human authors.

The **General Problem Solver** (1959) was a more ambitious follow-up, aiming to be a universal problem-solver not limited to the mathematical domain.

It introduced a key cognitive heuristic (*means-ends analysis*) to compare its current state to the desired goal, identify the differences, and apply operators to *reduce those differences*, creating sub-goals along the way.

It solved various well-defined problems, including the Towers of Hanoi puzzle and geometric proofs.

… After Thinking about This

Computing Machinery and Intelligence (1950)

Alan Turing recognised that the question "Can machines think?" was too vague and philosophical to be answered directly.

He proposed replacing it with a concrete, *operational test* that could be practically evaluated.

1. It sidestepped the impossible debate about "consciousness" and defined intelligence in terms of observable, intelligent *behaviour.*
2. It set a clear, albeit controversial, benchmark for the field: creating a machine whose conversational behaviour is indistinguishable from a human's.
3. Turing pre-emptively addressed and refuted many of the arguments against machine intelligence that are still debated today (consciousness, creativity, emotion, etc).

In Turing's Words

«We may hope that machines will eventually compete with men in all purely intellectual fields. But which are the best ones to start with? Even this is a difficult decision. Many people think that a very abstract activity, like the playing of chess, would be best. It can also be maintained that it is best to provide the machine with the best sense organs that money can buy, and then teach it to understand and speak English. This process could follow the normal teaching of a child. Things would be pointed out and named, etc. Again I do not know what the right answer is, but I think both approaches should be tried.»

Artificial Shrinks before GenAI

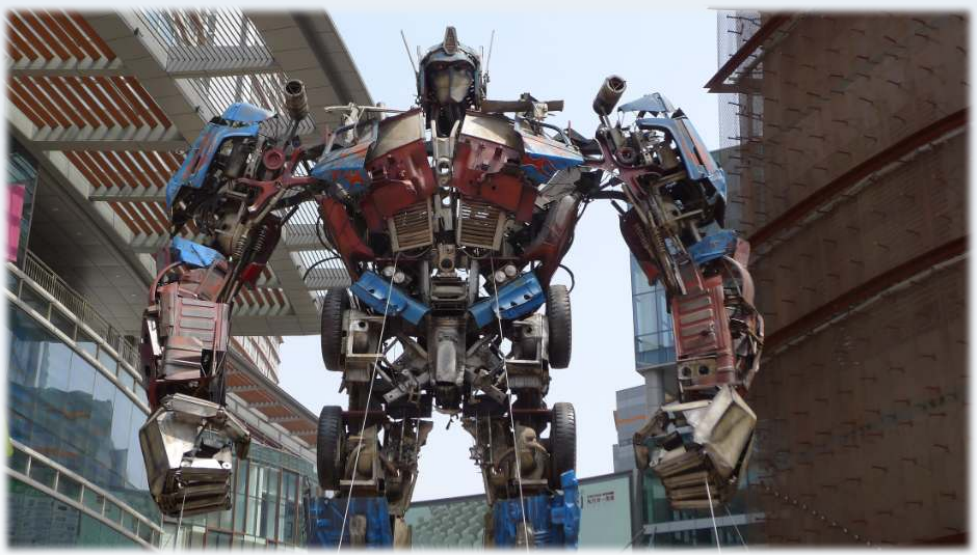
```
  Welcome to
                 EEEEEE  LL      IIII   ZZZZZZ   AAAAA
                 EE      LL       II       ZZ   AA   AA
                 EEEEE   LL       II     ZZZ    AAAAAAA
                 EE      LL       II    ZZ      AA   AA
                 EEEEEE  LLLLLL  IIII ZZZZZZ    AA   AA

  Eliza is a mock Rogerian psychotherapist.
  The original program was described by Joseph Weizenbaum in 1966.
  This implementation by Norbert Landsteiner 2005.


ELIZA: Is something troubling you ?
YOU:   Men are all alike.
ELIZA: What is the connection, do you suppose ?
YOU:   They're always bugging us about something or other.
ELIZA: Can you think of a specific example ?
YOU:   Well, my boyfriend made me come here.
ELIZA: Is it important to you that your boyfriend made you come here ?
YOU:   He says I'm depressed much of the time.
ELIZA: I am sorry to hear that you are depressed.
YOU:   It's true. I am unhappy.
ELIZA: Can you explain what made you unhappy ?
YOU:
```

An Euphoric Analogy

“AI in the ‘50s”

“Neural AI”

From Units to Transformers



Neural Networks

A computational model “inspired” by the human brain that learns to recognise patterns in data

Structure

- Interconnected nodes, or “neurons”, organised in layers.
- Data flows from an **input layer**, through one or more **hidden layers**, to an **output layer**.

Mechanism

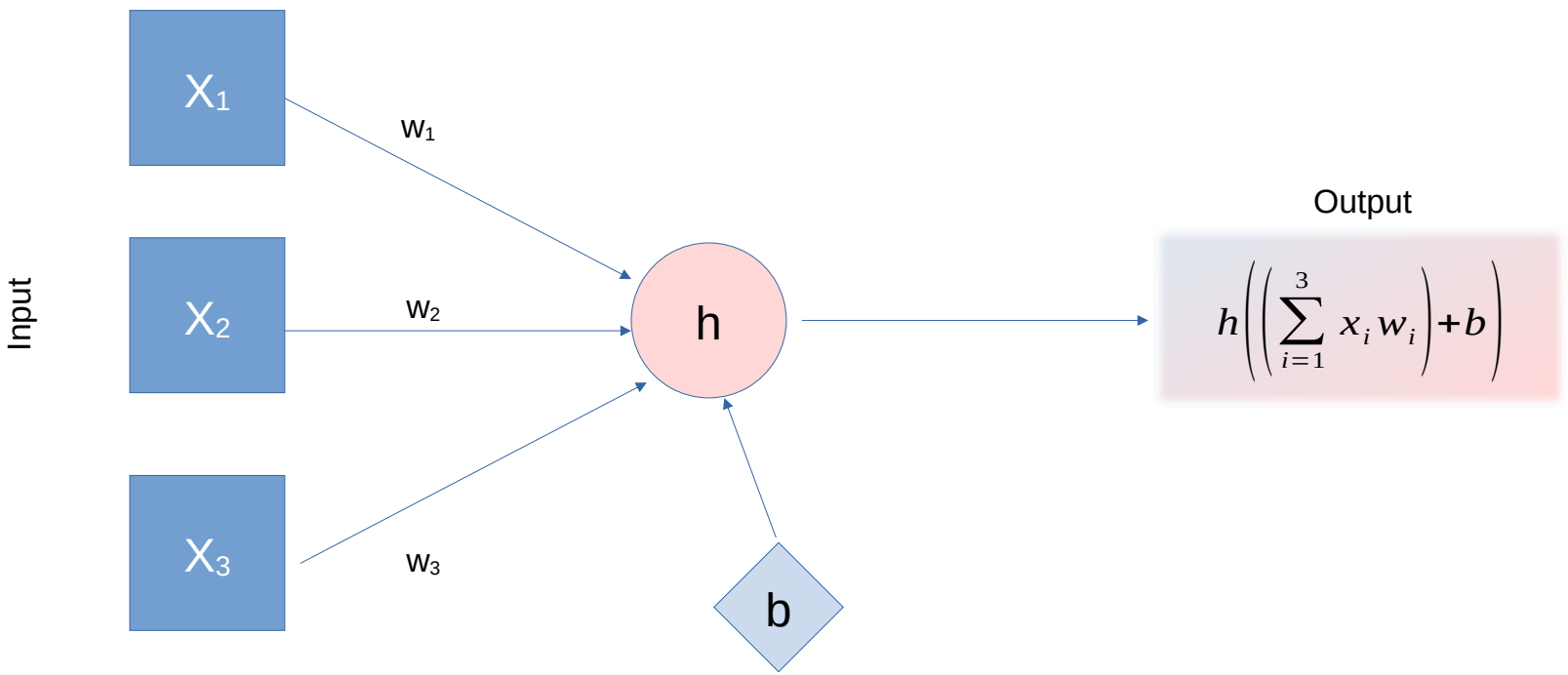
- The network “learns” by adjusting the strength of the connections between neurons.
- These connection strengths are represented by numerical values called **weights**.
- Each neuron also has an associated **bias**, which is another learnable parameter that helps **to control the neuron's output**.

Preprocessing

N.B. Raw data is rarely fed directly into a neural network: a crucial step is **feature scaling**, where input features are normalised

- A common method is to subtract the feature's mean and divide by its standard deviation across the training dataset.
- This helps the learning algorithm to converge more efficiently.

The Unit


Input
X1
X2
X3
w1
w2
w3
h
b
Output
$h\left(\left(\sum_{i=1}^{3} x_i w_i\right)+b\right)$

The “Firing” Mechanism of Neurons

Activation functions enable the network to learn complex patterns. Without them, a neural network would simply be a linear model.

ReLU: The most common activation function for hidden layers. It is defined as:

$$ReLu(x) := max\{x,0\}$$

Sigmoids: Often used in the output layer of a binary classification model.
They squash the input into a range between 0 and 1, which can be interpreted as a probability.

Leaky ReLU: A variant of ReLU that allows a small, non-zero gradient μ when the unit is not active.

$$LReLu(x) := \begin{cases} x, \ if \ x \geq 0 \\ \mu x, \ otherwise \end{cases}$$

The Learning Process

Training a neural network is an **iterative process of minimising the error** in its predictions.

1. **Initialisation:** The network's weights and biases are initialised with small *random values*.
2. **Forward Propagation:** A *batch of training data* is passed through the network to generate predictions.
3. **Loss Calculation:** A *loss function* measures the discrepancy between the network's predictions and the true labels in the training data.
4. **Back-propagation:** The error is propagated backward through the network, and the *gradient of the loss function* with respect to each weight and bias is calculated. This gradient indicates the direction of steepest ascent of the loss function.
5. **Parameter Update:** The *Gradient Descent* optimisation algorithm updates the weights and biases in the direction opposite to the gradient, thus taking a step towards minimising the loss.

Iteration: This process is repeated for many epochs (passes through the entire training dataset) until the *loss converges to a minimum*.

The Learning Engines

Two algorithms are the workhorses of neural network learning:

- **Back-propagation:** A highly efficient algorithm for *computing the gradients of the loss function with respect to the network's parameters*. It applies the chain rule of calculus to systematically calculate the gradients layer by layer, starting from the output layer and moving backward.

- **Gradient Descent:** An iterative optimisation algorithm used to find the minimum of a function. In the context of neural networks, *it uses the gradients computed by back-propagation to update the network's parameters in a way that minimises the loss function.* The size of the update step is controlled by a hyper-parameter called the learning rate.

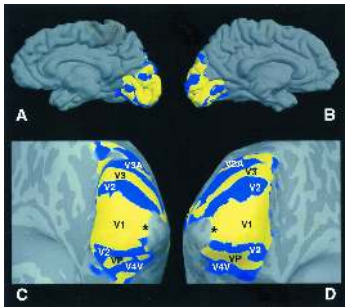
The Vision Specialists

Convolutional Neural Networks (CNNs) are the cornerstone of modern computer vision systems, since they are particularly well-suited for processing grid-like data, such as *images*.

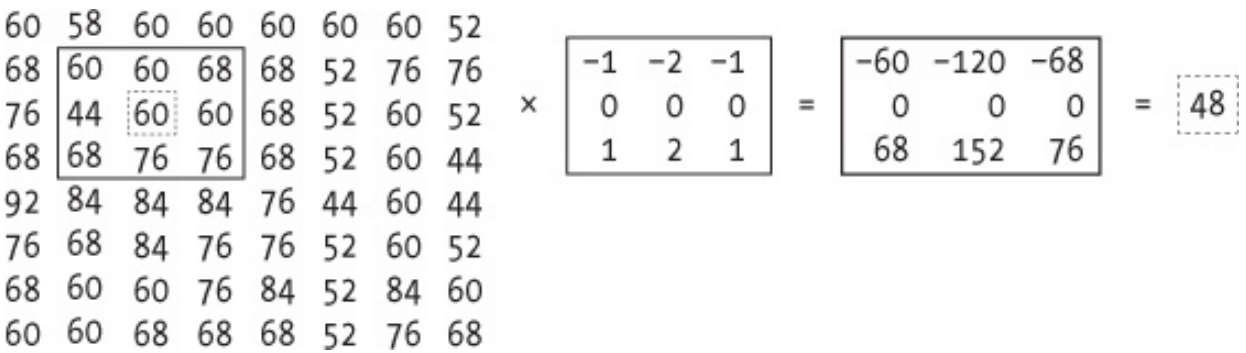
Key Innovation: CNNs are designed to automatically and adaptively *learn spatial hierarchies of features from input images*. This is achieved through the use of *convolutional layers*.

Applications: CNNs have achieved state-of-the-art performance on a wide range of computer vision tasks, including *image classification, object detection,* and *semantic segmentation*.

Inspiration: The architecture of CNNs is *inspired by* the organisation of *the biological visual cortex*.



Convolution



- Slide a small square, called a kernel, over the image.
- Movement occurs from top to bottom and left to right.
- At each position, pixel values within the kernel are multiplied by corresponding kernel values.
- The products are summed to produce a single output pixel value for that position.

Convolutional Architecture

CNNs are typically composed of three main types of layers:

Convolutional Layers apply a set of learnable filters (or *kernels*) to the input image. *Each filter is designed to detect a specific feature*, such as an edge, a corner, or a texture.

Pooling Layers are used to *downsample the spatial dimensions* of the feature maps, which reduces the number of parameters and computational complexity of the network, and also helps *to make the learned features more robust* to small translations and distortions in the input image.

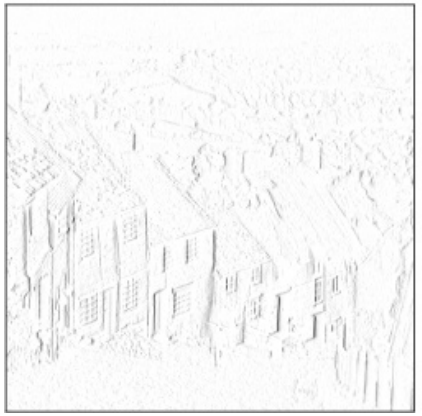
Dense Layers are typically found *at the end of the network* and are used *to perform the final classification task* based on the high-level features extracted by the convolutional and pooling layers.

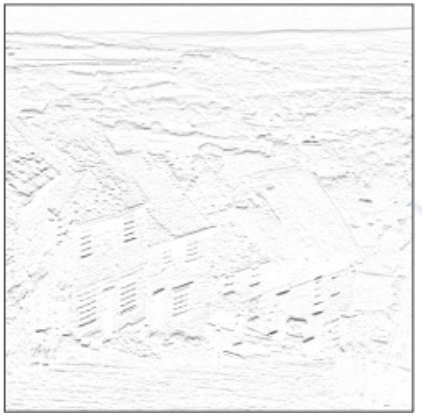
Gold Hill in Shaftesbury, England

Input/Model

Blurring kernel

Vertical kernel

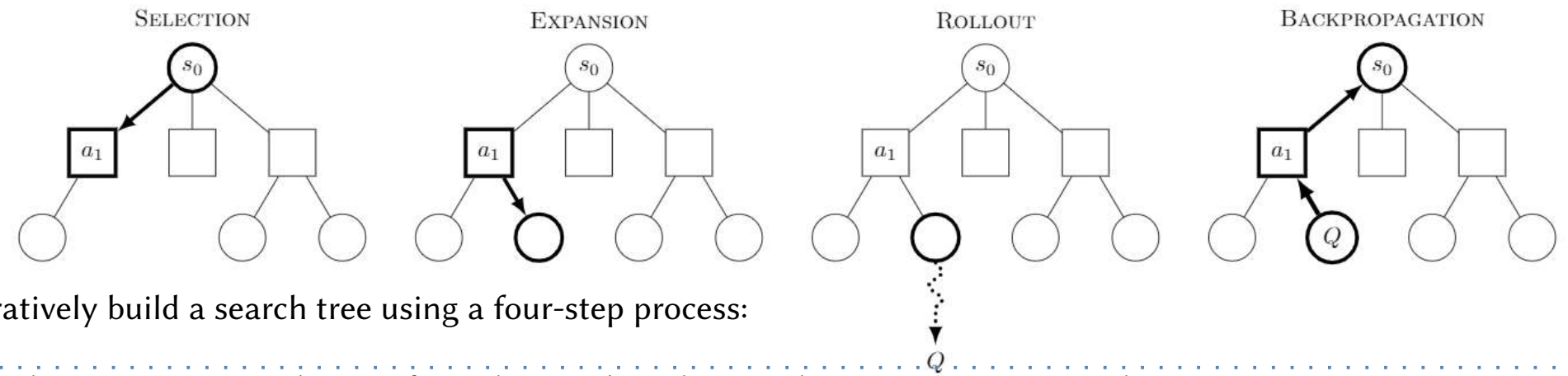




Horizontal kernel

Navigating Vast Search Spaces

Monte Carlo Tree Search is a powerful search algorithm for finding optimal decisions in domains with large and complex search spaces, such as board games (Go and Chess, for instance)



Iteratively build a search tree using a four-step process:

1. **Selection:** Traverse the tree from the root by selecting the most promising nodes.
2. **Expansion:** Add a new node to the tree.
3. **Rollout:** Run a random simulation from the new node to determine the outcome.
4. **Back-propagation:** Update the statistics of the nodes in the selected path based on the simulation result.

The *Upper Confidence Bound* for Trees (UCT) is a commonly used formula in the selection step *to balance exploration* (investigate new, unproven moves) *and exploitation* (focus on moves that have proven to be effective in the past).

The Power of Collaboration

The AlphaGo system by DeepMind (2016), achieved superhuman performance in the game of Go by combining MCTS with deep neural networks. It uses two neural networks to guide the MCTS:

I. A **Policy Network** is trained to predict the most likely moves for a human expert player, to narrow down the search space for MCTS.

II. A **Value Network** is trained to evaluate a given board position and estimate the probability of winning from that position.

Self-play Reinforcement Learning

AlphaZero (2018) is a more general-purpose algorithm that can learn to play multiple games (Go, Chess, and Shogi, for instance) at a superhuman level, *starting from scratch*.

Tabula Rasa Learning: Unlike AlphaGo, AlphaZero is *not trained on any human data*. It learns entirely through self-play reinforcement learning.

Unified Architecture: AlphaZero uses a *single, unified neural network* that takes the board position as input and outputs both move probabilities (policy) and a position evaluation (value).

The Art of Deception

Generative Adversarial Networks (GANs) are **generative models** that are capable of producing *highly realistic* synthetic data.

They consist of two neural networks, a Generator and a Discriminator, that are trained in an adversarial (competitive) process.

The **Generator** learns to create synthetic data that is indistinguishable from real data.

The **Discriminator** learns to distinguish between real data and the synthetic data produced by the generator.

The two networks are *trained simultaneously in a zero-sum game*, where the generator tries to fool the discriminator, and the discriminator tries to avoid being fooled.

The Adversarial Training Process

The Generator starts its training with a random vector of numbers called a *noise vector*: each unique noise vector forces the generator to produce a different image, allowing it to create a wide variety of outputs.
The training then proceeds in a two-phase loop:

Phase 1: Train the Discriminator. *The goal is to make the discriminator an expert at telling real from fake.*

1. The generator creates a batch of fake images from a set of noise vectors.
2. The discriminator is shown a mix of these fake images and a batch of real images from the training set.
3. The discriminator's weights are updated based on how well it distinguished the real images (the 1s) from the fake ones (the 0s).

Phase 2: Train the Generator. *The goal is to make the generator better at fooling the discriminator.*

1. The discriminator's weights are frozen so it provides a stable target for the generator.
2. The generator creates a new batch of fake images from different noise vectors.
3. These images are passed to the discriminator, and the generator's loss is calculated based on how close the discriminator's outputs were to 1 (real).
4. The generator's weights are updated to improve its ability to create images that fool the discriminator.

This cycle continues, with both networks getting progressively better until the generator's fakes are so good that the discriminator is no better than random chance at identifying them.

Control and Stability in GANs

Training GANs can be notoriously difficult and unstable. One of the most common challenges is

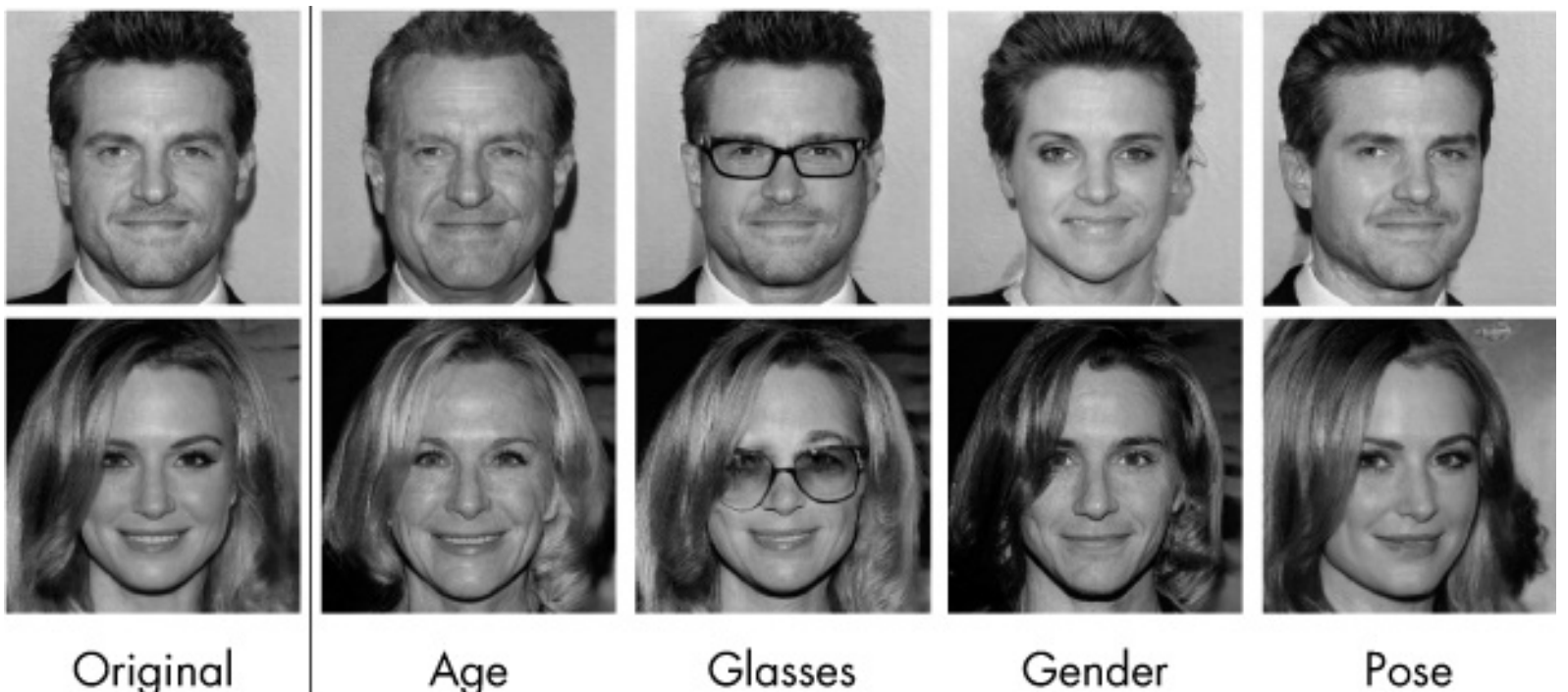
Mode Collapse, which occurs when the generator discovers a few "tricks" or outputs that can easily fool the discriminator. It then starts producing only this limited variety of outputs, regardless of the input noise vector. It then fails to capture the full diversity of the data.

Various techniques have been developed to mitigate it, for instance

In **Conditional GANs**, both the generator and the discriminator are conditioned on some extra information, such as a class label.

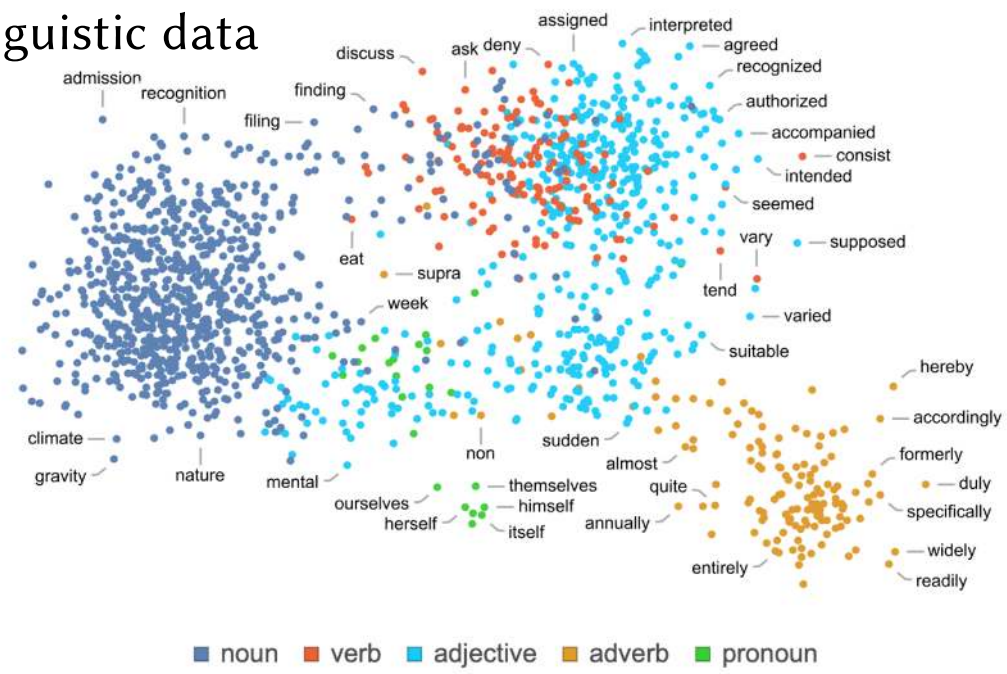
In **Controllable GANs**, more advanced GAN architectures allow for fine-grained control over the generated images by manipulating the *latent space*, which is a low-dimensional representation of the data that is learned by the generator. By moving in different directions in the latent space, it is possible to control specific attributes of the generated images.

Beyond Classical Face Recognition



How to Bake Your LLM

- *Huge* text corpora (mainly from the web) produced by humans
- *Vectorial encoding* of semantic proximity of text components
- Neural/*sub-symbolic* computational architecture
- *Matrix operations* on (encoding of) linguistic data
- GPUs used for *parallel computing*



The Recipe

These are statistical encodings of other words that are “called by” or “calling for” the given word

1. Take a string of linguistic items (say, words) in a given text
2. Assign to each of them 3 matrices: Q*ueries*; K*eys*; V*alues*
3. Pair Q and K matrices to compute probability weights
4. Generate average values using these probability weights and the V matrix
✔ Iterate, to complete BERT- and GPT-style tasks, modulo values refinements

Huge parallel computations (in GPUs) for searching syntactic structures within semantic encoding

Q and K matrices are a statistical proxy for syntactic relationship

Syntax is re-constructed from its semantic mirror

Previous Linguistic Models

Recurrent Neural Networks (RNNs) processed text sequentially, word by word.

They have a "memory" that allows them to use information from the prior input in the sequence to influence the current input and output.

When an RNN processes a sequence, it takes the first item (e.g., the first word of a sentence), performs a calculation, and produces an output. Crucially, it also passes a piece of information to the next step. When it processes the second word, it considers both the new word and the hidden state from the previous step.

The Bottleneck

An RNN tries to compress the meaning of an entire sentence or paragraph into a single vector of numbers (the "context").

For long sentences, information from the beginning is often lost or diluted by the time the model reaches the end. The model struggles to remember which words were most important.

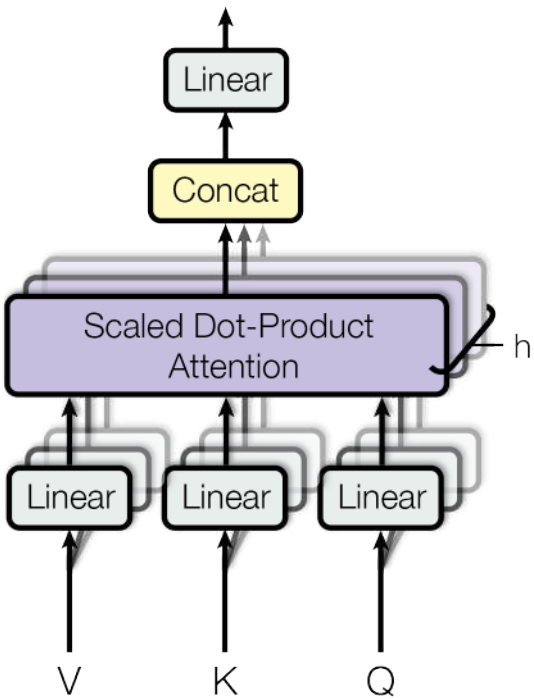
A Solution to “Forgetting”

Attention mechanisms allows models to *look back to the entire input sequence* at every step of the output generation.

Core Idea

Instead of relying on a single, compressed context vector, attention creates a direct shortcut to every input word via the QKV matrices.

The model selectively focuses on the most relevant parts of the input for the specific word it's currently generating.



Emerging Alternatives

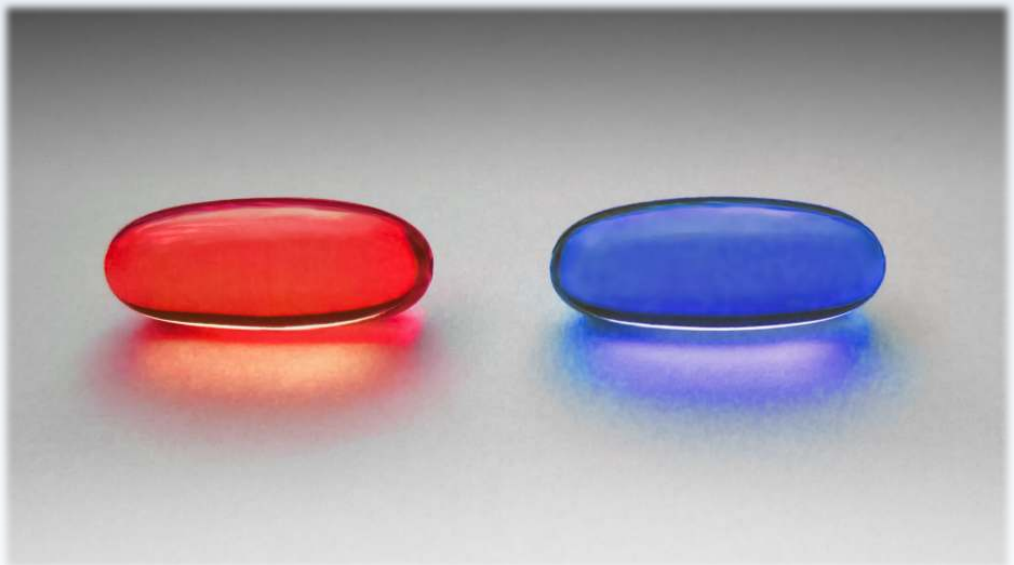
Attention-Free Transformers are a new class of models that aim to replace the expensive attention module with *more efficient alternatives*.

I. **State-Space Models** are inspired by signal processing, and use a continuous "state" to carry information along the sequence. This allows them to capture long-range dependencies with linear complexity.

II. **Gated Convolutions** use convolutional filters (like those in CNNs) combined with gating mechanisms to control the flow of information.

III. **Fourier Transformers** use (Fast) Fourier Transforms to mix information in the frequency domain, which is a mathematically efficient way to combine signals from the entire sequence at once.

How Deep the Rabbit Hole Goes



The “Fravico” Explanation

- (noun): a feeling of relief when you find something you have been looking for for a long time.

  "He felt an immense fravico finding the keys under the sofa."

- (adjective): said of an object that is apparently solid, but actually fragile.

  "The chair looked sturdy, but it was so fravica that it broke immediately."

- (verb): to conclude a discussion or negotiation suddenly.

  "He managed to fravicare the contract with a single sentence."

- (noun): a tropical plant used to relieve headaches.

  "Fravico tea is a remedy for migraines."

The Unexpected Roots of LLMs

Zellig Harris' linguistic theory (1940s-50s):

- The *Distributional Hypothesis*:
  - Language as a set of text strings.
    - The meaning and grammatical function of a word are determined by the contexts in which it appears.
    - Syntax is seen as a set of probabilistic relationships between words, not as an abstract, internal structure.

A Modern Paradigm

Noam Chomsky's generative linguistics (1957 –):

- The *Minimalist Program (1990s –)*:
  - An underlying *simple* computational structure is responsible for linguistic capability in the *human* mind ("Universal Grammar").
    - The central operation of this grammar is a *simple*, *recursive function* called Merge.
    - Syntax/Language is fundamentally about building *hierarchical structures* by merging recursively two linguistic objects, while *minimising the computational costs* of the operation.

Impossible Languages

The Test:

Participants were taught two types of artificial languages while their brains were scanned:

- "Possible" Language: Used *hierarchical, structure-dependent rules*
  E.g., «move the verb from the main clause».
- "Impossible" Language: Used (computationally simple but) unnatural *linear rules* that no human language follows
  E.g., «move the third word to the front».

The Results:

- Learning the "possible" language activated Broca's area: the brain recognised it as a real language and tried to learn it.
- Learning the "impossible" language did NOT activate Broca's area: the brain treated it like a logic puzzle or a math problem, using different cognitive regions, but it did not process it as language.

Cultural Convention vs Biological Necessity

Andrea Moro's experiment shows the human brain is not a general-purpose learner; it has a specific, *biological bias* for hierarchical structures.

Mission: Impossible Languages (arXiv 2024)

Models like GPT-2 do not learn all patterns equally well:

- "Natural"/hierarchically structured languages learnt more efficiently and with lower error rates than "impossible" languages.

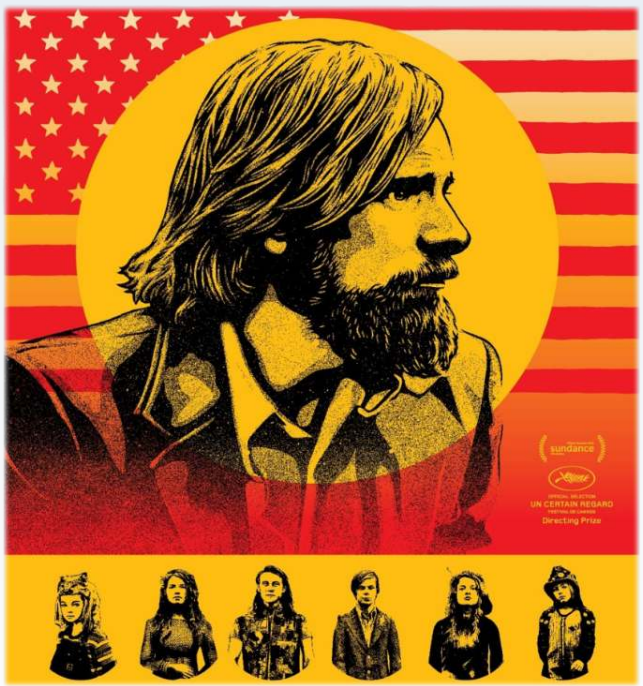
Systematic testing of three Language Models reveals low language accuracy, absence of response stability, and a yes-response bias (PNAS 2023)

- LLMs perform nearly at chance in some language judgment tasks, while revealing a stark absence of response stability and a bias toward yes-responses.

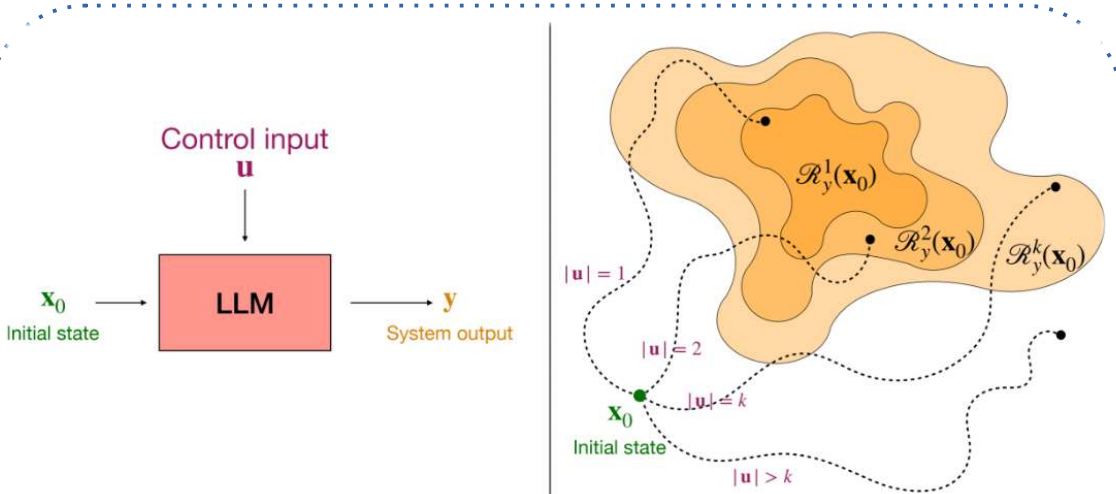
Intrinsic Dimension Estimation for Robust Detection of AI-Generated Texts (NeurIPS 2023)

- The average intrinsic dimensionality of AI-generated texts for each language is ≈1.5 lower than of fluent texts in a natural language.

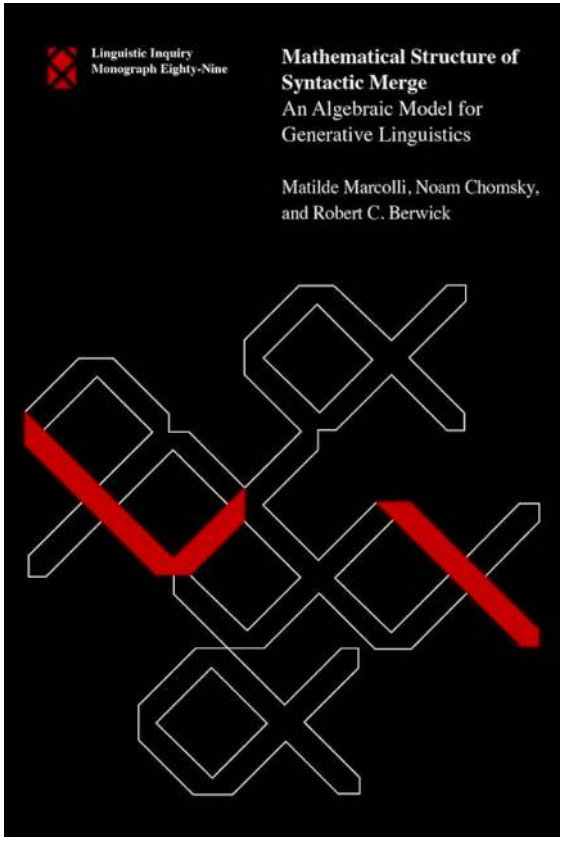
Conclusions



«If you assume that there is no hope...»



What's the Magic Word? A Control Theory of LLM Prompting (arXiv 2023)



"Strong Minimalist Thesis + Maths" (MIT Press 2025)

(Kind of) Take Home

LLMs *excel at prediction*: They are incredibly powerful *engineering achievements* that create fluent and coherent text by mastering the statistical patterns of language. However, this predictive power does not, in itself, constitute a scientific explanation.

For having a *genuine* (linguistic) AI we need to provide a concise, testable, and falsifiable conceptual model of how language works.
Mechanistic interpretability is actively trying to reverse-engineer LLMs to extract the "algorithms" they have learned. To date, however, these efforts have not revealed a system that looks like the rules of generative syntax.

Pure "data + technology", no matter how powerful, *do not constitute* science and *conceptual progress* on their own.

A Tenuous Analogy

"AI in the '50s"

"Generative *Supposedly* AI"

A More Correct Analogy

"AI in the '50s"

"Neural Generative AI"

An Optimistic Analogy

"AI in the '50s"

"*Genuine* AI"

IMT SCUOLA ALTI STUDI LUCCA

Thank You for Listening!

cosimo.perinibrogi@imtlucca.it