AI for Mathematics

AI Foundations → Mathematical Research Projects

Ma Jiajun

School of Math, XMU / Dept. of Math, XMUM

April 28 & May 5, 2026

Part I

The Logic and History of AI Research Methods

What Is Artificial Intelligence?

Artificial intelligence is the broad field of building systems that perform tasks associated with intelligence.

Reasoning

Make inferences from given information.

Learning

Improve behavior from data or feedback.

Language

Understand or generate text and speech.

Perception

Interpret images, sound, or sensor data.

Planning

Choose actions to reach a goal.

Problem solving

Search for solutions under constraints.

Reference: S. Russell and P. Norvig, *Artificial Intelligence: A Modern Approach*, 4th ed., 2021, Ch. 1.

AI Is Broader Than LLMs

Correct picture

LLMs are one recent and important family of AI systems.

Wrong shortcut

"AI" does not mean only ChatGPT-style systems.

In this course, LLMs matter as one interface to language, code, and tool workflows.

References: S. Russell and P. Norvig, *Artificial Intelligence: A Modern Approach*, 2021; R. Bommasani et al., "On the Opportunities and Risks of Foundation Models", 2021.

Vocabulary Map

AI: broad field of machine intelligence

Machine learning: behavior learned from data

Deep learning: machine learning with large multi-layer neural networks

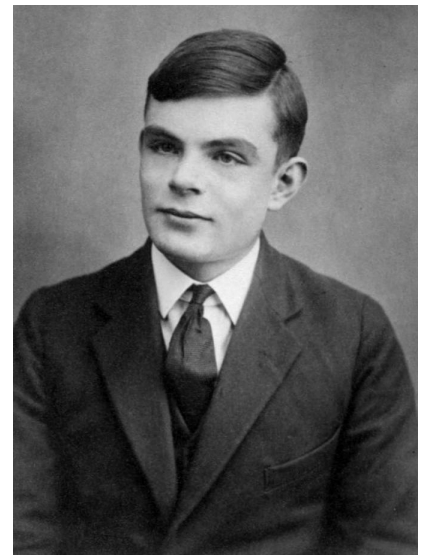
Generative AI: systems that generate text, images, audio, code, or other content

LLM: generative AI model specialized in language and code-like token sequences

References: I. Goodfellow, Y. Bengio, A. Courville, *Deep Learning*, MIT Press, 2016; C. M. Bishop and H. Bishop, *Deep Learning: Foundations and Concepts*, Springer, 2024, Ch. 12.

Historical Timeline

Period	Main idea
1950	Turing's imitation game
1955/1956	Dartmouth proposal/workshop
1950s-1970s	Symbolic AI, search, planning
1980s	Expert systems
1990s-2000s	Statistical machine learning
2010s	Deep learning at scale
2017	Transformer architecture
2020s	LLMs and tool-using agents



Alan Turing's 1950 paper made machine intelligence a concrete scientific question.

References: A. M. Turing, "Computing Machinery and Intelligence", *Mind*, 1950; J. McCarthy, M. L. Minsky, N. Rochester, C. E. Shannon, Dartmouth AI proposal, 1955; A. Vaswani et al., "Attention Is All You Need", 2017.

Two Long-Running Impulses

Symbolic

Represent knowledge explicitly with rules, logic, search, and plans.

Statistical

Learn behavior from data using probability, optimization, and models.

Modern systems often combine both: neural models plus search, retrieval, code execution, or proof checking.

References: A. Newell and H. A. Simon, "Computer Science as Empirical Inquiry: Symbols and Search", 1976; J. Pearl, *Probabilistic Reasoning in Intelligent Systems*, 1988.

Method 1: Symbolic AI / GOFAI

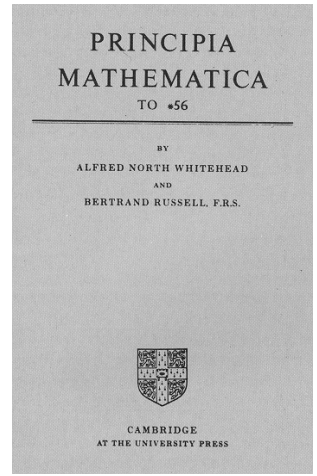
Core idea: represent knowledge with symbols, rules, and logic.

Strength

Transparent reasoning; good when rules are explicit and the domain is formal.

Limitation

Brittle when the world is messy, ambiguous, or too large to encode by hand.



Symbolic AI inherited confidence from logic, formal systems, and explicit rules.

Reference: A. Newell and H. A. Simon, "Computer Science as Empirical Inquiry: Symbols and Search", ACM Turing Award lecture, 1976.

Example: Logic Theorist

Background

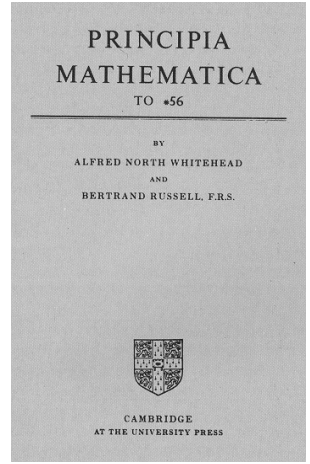
Logic Theorist was built by Allen Newell, Herbert Simon, and Cliff Shaw in 1956, at the beginning of AI as a named research field.

What it did

It searched for proofs of propositional logic theorems from *Principia Mathematica*. The task was narrow, formal, and rule-governed.

Why it matters

It made the early AI dream concrete: intelligence as explicit symbol manipulation plus heuristic search.



Formal proof was one of AI's earliest test domains.

References: A. Newell, J. C. Shaw, H. A. Simon, *The Logic Theory Machine*, 1956/1957; L. Gugerty, "Newell and Simon's Logic Theorist: Historical Background and Impact on Cognitive Modeling", 2006.

Method 2: Search

Core idea: formulate a task as states and actions, then explore possible paths until a goal is reached.

Component	Meaning
State	a compact description of where we are
Action	one legal move from a state
Successor function	generates next states
Goal test	decides whether the task is solved
Search strategy	chooses which unfinished node to expand

A search algorithm maintains a frontier of unfinished nodes. The strategy is what makes breadth-first, depth-first, best-first, and A* different.

References: P. E. Hart, N. J. Nilsson, B. Raphael, "A Formal Basis for the Heuristic Determination of Minimum Cost Paths", 1968; S. Russell and P. Norvig, *Artificial Intelligence: A Modern Approach*, 2021, Chs. 3-4.

Tree Search

Node

A node records a state plus the path that reached it. The same state can appear in more than one path unless duplicates are detected.

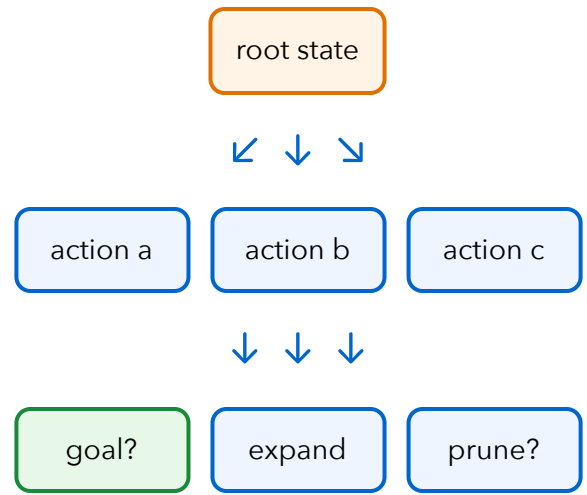
Expansion

To expand a node, apply each legal action and create child nodes.

Frontier

The frontier is the set of generated but not-yet-expanded nodes. Search is mostly the problem of choosing from this frontier.

A search tree



The tree is a bookkeeping structure for possible histories, not necessarily a picture of the whole world.

References: S. Russell and P. Norvig, *Artificial Intelligence: A Modern Approach*, 4th ed., Ch. 3; P. E. Hart, N. J. Nilsson, B. Raphael, "A Formal Basis for the Heuristic Determination of Minimum Cost Paths", 1968.

Proof Search as Tree Search

Search object

In theorem proving, a node can be a proof state: current hypotheses plus goals still to prove.

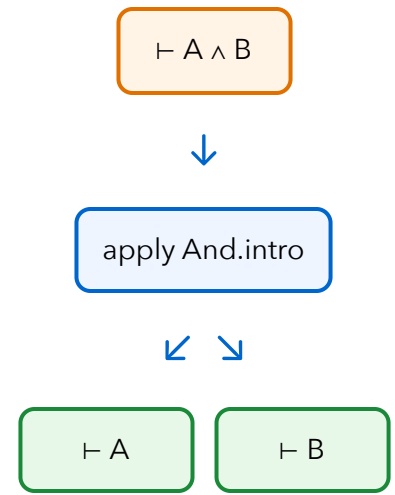
Edges

An inference rule or tactic transforms one goal into zero or more subgoals.

Success condition

A branch succeeds only when every leaf goal is closed by an assumption, axiom, lemma, computation, or decision procedure.

Example proof tree



Lean tactics expose this process interactively; automated provers search much larger proof spaces.

References: J. A. Robinson, "A Machine-Oriented Logic Based on the Resolution Principle", *Journal of the ACM*, 1965; J. Avigad et al., Theorem Proving in Lean 4, Ch. 5: Tactics.

Example: Deep Blue

Background

IBM's Deep Blue defeated world chess champion Garry Kasparov in a 1997 match.

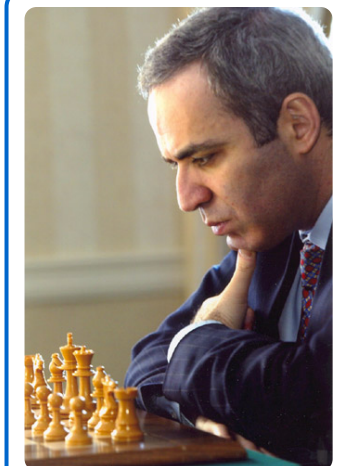
What made it work

It searched enormous chess game trees, evaluated board positions, and used specialized hardware plus chess knowledge.

Method point

Search is powerful when rules are fixed, actions are discrete, and states can be evaluated.

IBM Deep Blue chess computer.



Garry Kasparov, world chess champion.

The match is a classic example of search, evaluation, and domain engineering.

References: IBM, "Deep Blue", official history page; M. Campbell, A. J. Hoane Jr., F.-h. Hsu, "Deep Blue", *Artificial Intelligence*, 2002.

Method 3: Planning

Planning asks for a legal action sequence, not just a plausible next step.

Input

An initial state, a goal condition, and a model of available actions.

Action model

Each action specifies when it is allowed and how it changes the state.

Output

A sequence $a_1, \ldots, a_T$ such that every precondition is satisfied and the final state satisfies the goal.

STRIPS-style action

preconditions: facts that must already hold

add list: facts made true by the action

delete list: facts made false by the action

State update:

$$\text{new state} = (\text{old state} - \text{delete}) + \text{add}$$

Planning is search over action sequences plus legality checks at each step.

Reference: R. E. Fikes and N. J. Nilsson, "STRIPS: A New Approach to the Application of Theorem Proving to Problem Solving", *Artificial Intelligence*, 1971.

Example: Shakey and STRIPS

Background

Shakey was an early mobile robot project at SRI in the late 1960s and early 1970s.

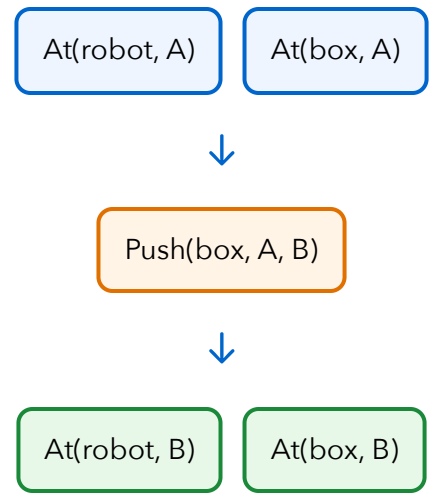
What the planner had to do

Represent facts about rooms, doors, the robot, and boxes; then choose actions whose preconditions become true in order.

Why this is planning

A command such as "put the box in the target room" is not one action. It must be decomposed into moves, pushes, and checks.

Toy STRIPS action



The plan is valid only if every action's preconditions hold at that step.

References: SRI International, "Shakey the Robot"; R. E. Fikes and N. J. Nilsson, "STRIPS: A New Approach to the Application of Theorem Proving to Problem Solving", *Artificial Intelligence*, 1971.

Method 4: Knowledge-Based / Expert Systems

Core idea: answer questions or make decisions by combining stored knowledge with rules, retrieval, and scoring.

Classic expert systems

Hand-written rules work best in narrow domains with stable concepts and expert heuristics.

Modern knowledge systems

Question-answering systems often mix knowledge bases, search, NLP, and statistical confidence.

References: E. A. Feigenbaum, "The Art of Artificial Intelligence: Themes and Case Studies of Knowledge Engineering", 1977; D. Ferrucci et al., "Building Watson", 2010.

Example: IBM Watson on Jeopardy!

Background

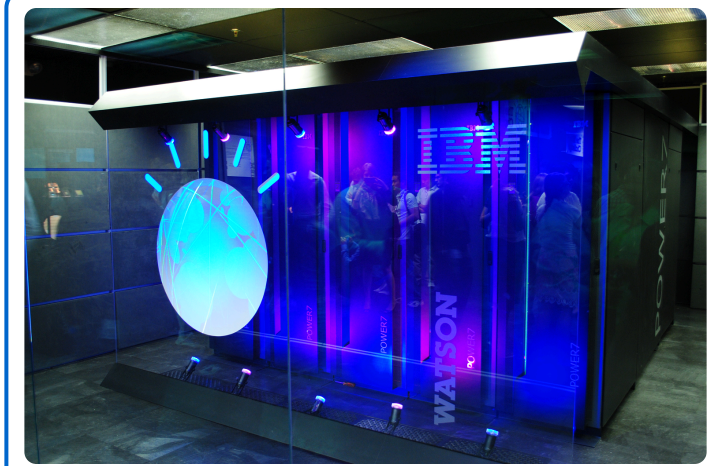
In 2011, IBM Watson defeated Jeopardy! champions Ken Jennings and Brad Rutter.

How it worked

It parsed clues, searched large knowledge sources, generated candidate answers, and ranked them by confidence.

System point

This is not a pure rule-based expert system: it shows how knowledge, language processing, retrieval, and statistical scoring can be combined.



IBM Watson prototype hardware. Photo: Clockready / Wikimedia Commons.

Pipeline: clue parsing → evidence retrieval → candidate answers → confidence ranking.

Reference: D. Ferrucci et al., "Building Watson: An Overview of the DeepQA Project", IBM Research / *AI Magazine*, 2010.

Method 5: Probabilistic AI

Core idea: represent uncertainty explicitly, then update probabilities when evidence arrives.

Hidden variables

The system may care about causes that are not directly observed.

Noisy evidence

Tests, sensors, and reports can be incomplete or wrong.

Posterior belief

Inference computes probabilities after conditioning on observations.

The output is often not "yes" or "no", but a distribution such as p(cause | evidence).

References: J. Pearl, *Probabilistic Reasoning in Intelligent Systems*, 1988; C. M. Bishop and H. Bishop, *Deep Learning: Foundations and Concepts*, Springer, 2024, Ch. 11.1.

Example: Bayesian Network for Diagnosis

Graph

Nodes are random variables. Arrows show which variables directly affect the local conditional probabilities.

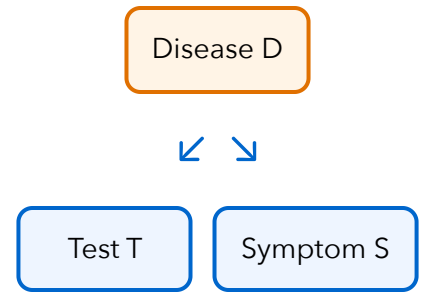
Factorization

For Disease D, Test T, and Symptom S, the graph below represents $p(D,T,S)=p(D)p(T|D)p(S|D)$.

Inference question

After observing T=positive and S=yes, compute $p(D \mid T=\text{positive}, S=\text{yes})$.

A small Bayesian network



In this simplified model, T and S are conditionally independent given D.

References: J. Pearl, *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*, Morgan Kaufmann, 1988; C. M. Bishop and H. Bishop, *Deep Learning: Foundations and Concepts*, Springer, 2024, Ch. 11.1, especially Eq. 11.6 for directed graphical model factorization.

Method 6: Statistical Machine Learning

Core idea: learn patterns from data using statistics and optimization.

This shifts the problem from writing all rules to choosing data, models, objectives, and evaluations.

Reference: C. M. Bishop, *Pattern Recognition and Machine Learning*, Springer, 2006, Ch. 1.

Example: Spam Filters

Background

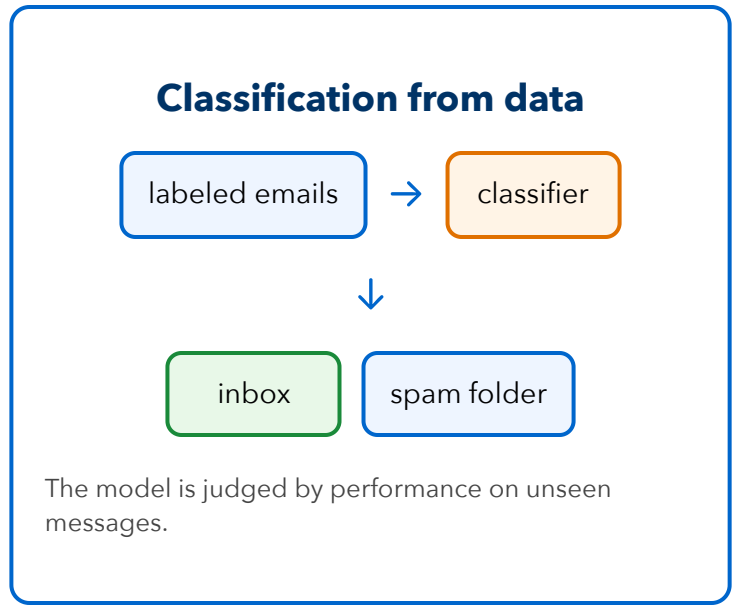
Email spam filtering was a practical success story for statistical machine learning in the 1990s and 2000s.

How it worked

Systems learned from labeled emails. Words, links, headers, and patterns became features for classifying messages.

Workflow

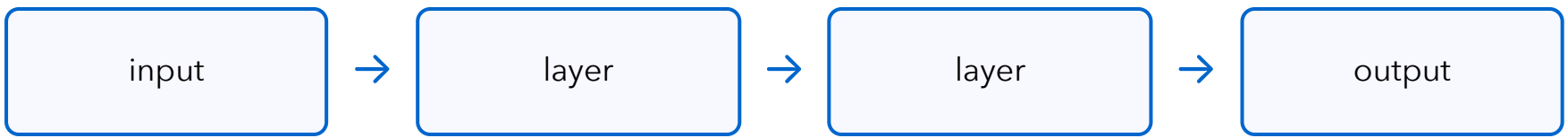
The workflow is empirical: collect examples, train a classifier, test on new examples, and update as data changes.



Reference: M. Sahami, S. Dumais, D. Heckerman, E. Horvitz, "A Bayesian Approach to Filtering Junk E-Mail", AAAI Workshop on Learning for Text Categorization, 1998.

Method 7: Connectionism and Neural Networks

Core idea: learn distributed representations with many parameters.



Historically powerful but hard to train at scale until better data, hardware, and techniques arrived.

References: W. S. McCulloch and W. Pitts, "A Logical Calculus of the Ideas Immanent in Nervous Activity", 1943; D. E. Rumelhart, G. E. Hinton, R. J. Williams, "Learning representations by back-propagating errors", 1986.

Example: LeNet-5

Background

LeNet-5, developed by Yann LeCun and collaborators in the 1990s, was an influential CNN for handwritten digit recognition.

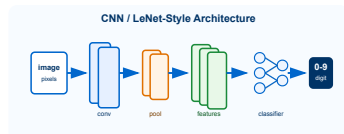
What it showed

Convolutions reuse small filters across an image, so the model can learn edges, strokes, and digit parts.

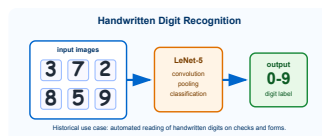
Architecture point

CNNs made neural networks practical for visual recognition before the deep learning wave of the 2010s.

Yann LeCun. Photo: Rama / Wikimedia Commons.



CNNs learn feature maps before classification.

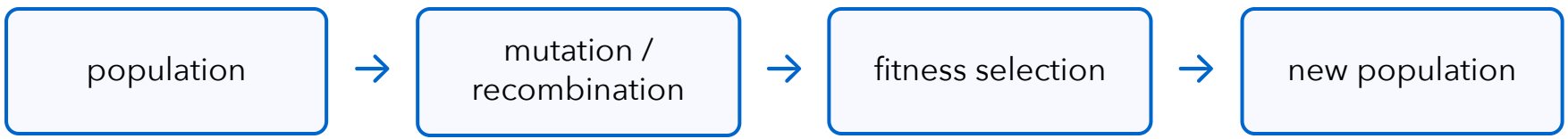


LeNet-style systems were built for handwritten digit recognition.

Reference: Y. LeCun, L. Bottou, Y. Bengio, P. Haffner, "Gradient-Based Learning Applied to Document Recognition", *Proceedings of the IEEE*, 1998.

Method 8: Evolutionary Computation

Core idea: search by variation and selection.



Useful for optimization and design search; often sample-inefficient and weak on guarantees.

References: J. H. Holland, *Adaptation in Natural and Artificial Systems*, 1975; K. A. De Jong, "Evolutionary Computation: A Unified Approach", 2006.

Example: NASA Evolved Antenna

Background

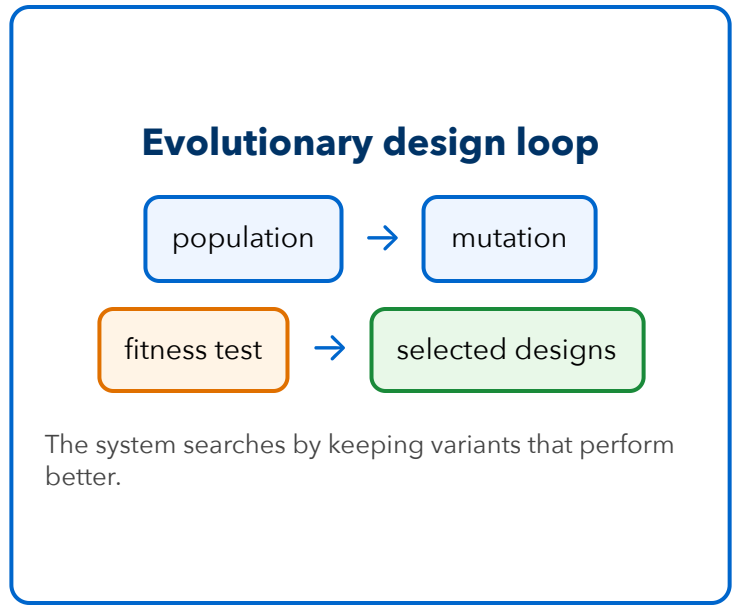
NASA used evolutionary design methods to produce unusual antenna geometries for space applications.

How it worked

Candidate antenna designs were mutated and selected according to performance criteria. Good designs seeded the next generation.

Search point

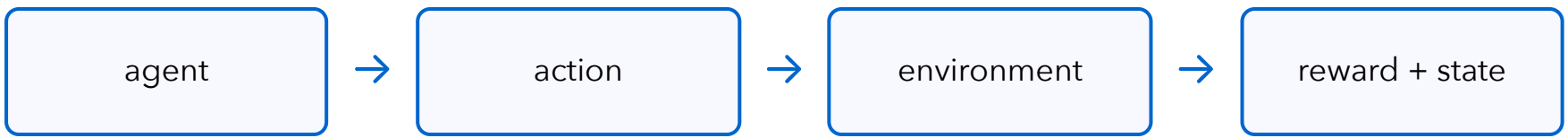
Evolutionary computation helps when the design space is strange and human intuition may not suggest the best shape.



Reference: G. S. Hornby et al., "An Evolved Antenna for Deployment on NASA's ST5 Mission", NASA Technical Reports Server, 2004.

Method 9: Reinforcement Learning

Core idea: learn actions from reward feedback in an environment.



Hard parts: exploration, reward design, and credit assignment over long action sequences.

Reference: R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*, 2nd ed., 2018.

Example: AlphaGo

Background

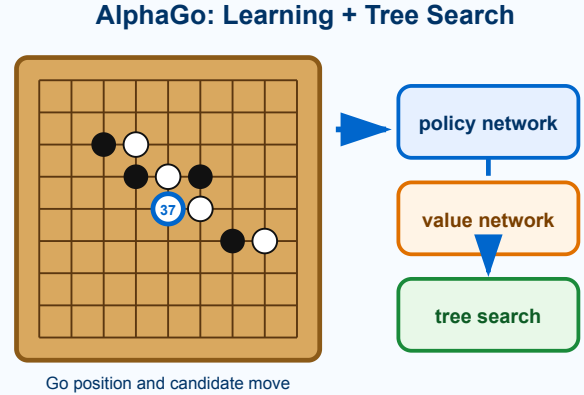
DeepMind's AlphaGo defeated Lee Sedol in 2016, a landmark result because Go has an enormous search space.

What it combined

AlphaGo used deep neural networks, reinforcement learning, and tree search. It learned from games and improved by self-play.

System point

Reinforcement learning is strongest when feedback from actions can become long-term improvement.



AlphaGo combines learned policy/value networks with tree search.

Famous context: AlphaGo defeated Lee Sedol 4-1 in Seoul in March 2016.

Reference: D. Silver et al., "Mastering the game of Go with deep neural networks and tree search", *Nature*, 2016.

Method 10: Deep Learning

Core idea: train large multi-layer neural networks on large datasets with gradient-based optimization.

Vision

Image classification, detection, generation.

Speech

Recognition and synthesis.

Language

Translation, summarization, generation.

Deep learning can fail silently: confident output is not the same as correct output.

References: Y. LeCun, Y. Bengio, G. Hinton, "Deep learning", *Nature*, 2015; C. M. Bishop and H. Bishop, *Deep Learning: Foundations and Concepts*, 2024.

Example: AlexNet

Background

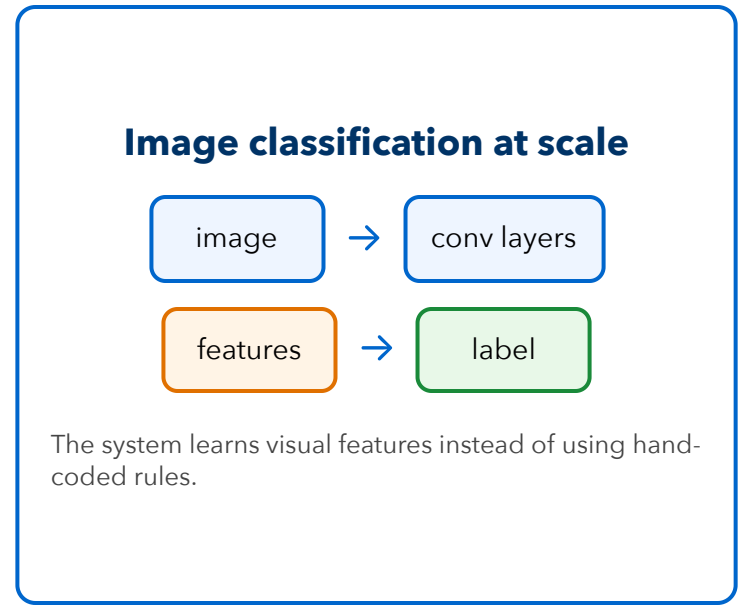
AlexNet won the 2012 ImageNet competition by a large margin and made deep convolutional networks impossible to ignore.

Why it mattered

It benefited from large labeled image datasets, GPUs, and architectural choices that made deep training practical.

Scaling point

Deep learning breakthroughs often come from combining data, compute, architecture, and optimization.

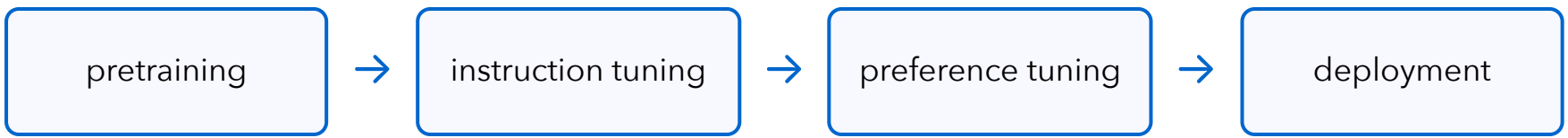


The system learns visual features instead of using hand-coded rules.

Reference: A. Krizhevsky, I. Sutskever, G. E. Hinton, "ImageNet Classification with Deep Convolutional Neural Networks", NeurIPS, 2012.

Method 11: Foundation Models and LLMs

Core idea: pretrain large models on broad corpora, then adapt them to many downstream tasks.



References: R. Bommasani et al., "On the Opportunities and Risks of Foundation Models", 2021; T. Brown et al., "Language Models are Few-Shot Learners", 2020.

Example: GPT-3 and ChatGPT

Background

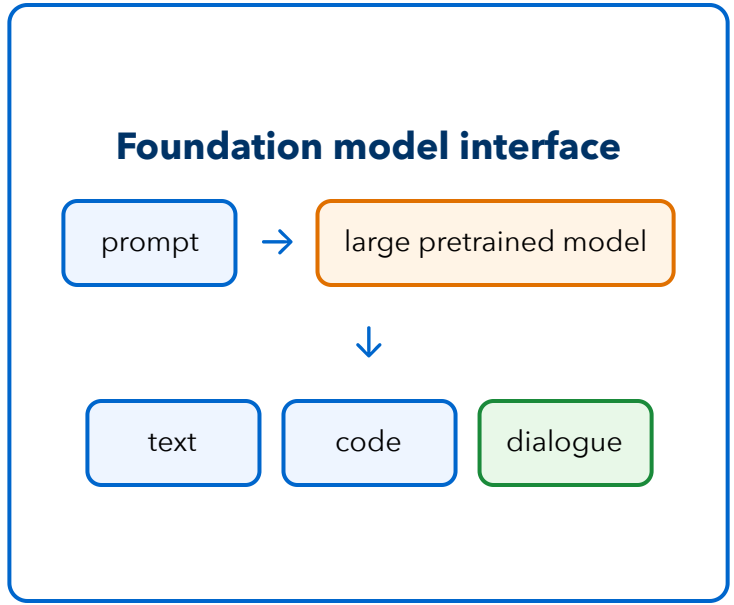
GPT-3 showed that scaling Transformer language models could produce broad few-shot text behavior. OpenAI released ChatGPT publicly in November 2022 as a dialogue interface for instruction-following models.

What changed

The same model interface could draft text, write code, explain concepts, translate, summarize, and interact in natural language.

Interface point

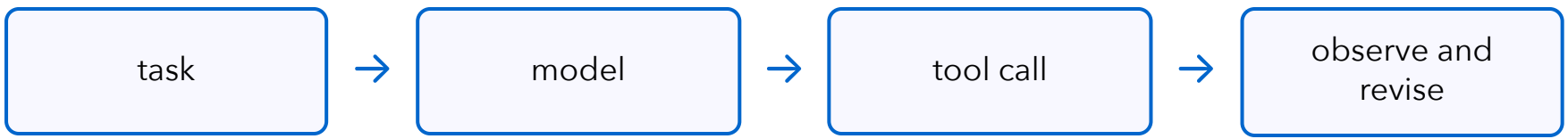
Foundation models are general-purpose interfaces, but fluency is not guaranteed correctness.



References: T. Brown et al., "Language Models are Few-Shot Learners", 2020; OpenAI, "Introducing ChatGPT", November 30, 2022.

Method 12: Agentic and Tool-Using AI

Core idea: put a model inside a loop with tools, memory, retrieval, and evaluation.



Agent systems will appear later when projects need tool use, logs, and external verification.

References: S. Yao et al., "ReAct: Synergizing Reasoning and Acting in Language Models", 2022; Anthropic, Claude Code overview.

Example: Claude Code-Style Coding Agent

Background

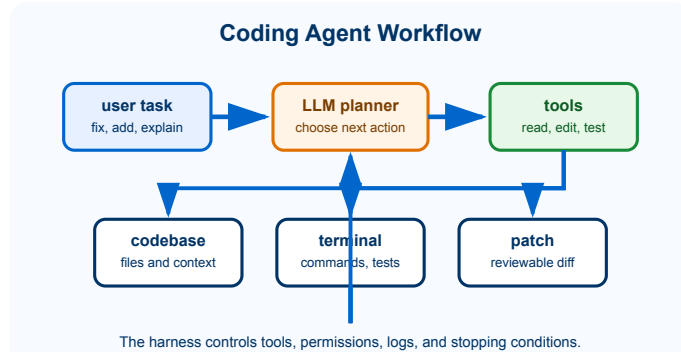
Coding agents such as Claude Code put an LLM inside a development harness that can inspect a repository and use tools.

What changes

The model can read files, search the codebase, edit patches, run commands, inspect errors, and iterate toward a working result.

Agent pattern

ReAct is a classic pattern for reason-plus-action loops; coding agents make the same idea concrete in a real workspace.



A coding agent is defined by its model, harness, tools, permissions, and logs.

References: Anthropic, Claude Code overview; S. Yao et al., "ReAct: Synergizing Reasoning and Acting in Language Models", 2022.

Sources: AI History Examples

Claim area	Sources used
Turing and Dartmouth	Turing, "Computing Machinery and Intelligence" (1950); McCarthy et al., Dartmouth proposal (1955)
Logic Theorist	Newell, Shaw, and Simon, *The Logic Theory Machine* (1956/1957); historical account: Gugerty (2006), "Newell and Simon's Logic Theorist"
Shakey / STRIPS	SRI, Shakey the Robot; Fikes and Nilsson, "STRIPS" (1971)
Deep Blue	IBM, Deep Blue; Campbell, Hoane, and Hsu, "Deep Blue" (2002)
Watson	Ferrucci et al., Building Watson: An Overview of the DeepQA Project
Bayesian networks and spam filtering	Pearl, *Probabilistic Reasoning in Intelligent Systems* (1988); Sahami et al., A Bayesian Approach to Filtering Junk E-Mail (1998)

Sources: Learning, Games, and Agents

Claim area	Sources used
LeNet-5	LeCun et al., Gradient-Based Learning Applied to Document Recognition (1998)
AlexNet	Krizhevsky, Sutskever, and Hinton, ImageNet Classification with Deep Convolutional Neural Networks (2012)
AlphaGo	Silver et al., Mastering the game of Go with deep neural networks and tree search (2016)
NASA evolved antenna	NASA NTRS, An Evolved Antenna for Deployment on NASA's ST5 Mission
GPT-3 / ChatGPT	Brown et al., Language Models are Few-Shot Learners (2020); OpenAI, Introducing ChatGPT
ReAct / coding agents	Yao et al., ReAct (2022); Anthropic, Claude Code overview

Part II

Language Data, Tokens, and Pretraining

The LLM Training Story

> A modern LLM is first trained as a probabilistic sequence model, then often post-trained to behave like an assistant.

Stage	Main object	What is optimized
Tokenization	text -> token IDs	fixed tokenizer chosen before model training
Embedding	token ID -> vector	embedding table learned with the model
Pretraining	Transformer language model	next-token likelihood / cross-entropy
Post-training	assistant behavior	demonstration loss and preference/RL objectives
Decoding	probability -> generated text	inference rule: greedy, sampling, top-p, temperature

References: T. Brown et al., "Language Models are Few-Shot Learners", 2020, Secs. 2.1-2.2; C. M. Bishop and H. Bishop, *Deep Learning: Foundations and Concepts*, Springer, 2024, Chs. 12.2-12.3.

What an LLM Learns From

> The training data is text. The model learns statistical structure that helps predict continuations in that text.

Published example	Data mixture described in the paper
GPT-3	filtered Common Crawl, WebText2, Books1, Books2, Wikipedia
Preprocessing	quality filtering, deduplication, and tokenization before training
Consequence	learned behavior reflects patterns and errors in the training distribution

> Pretraining is not the same as training on a database of verified propositions.

Reference: T. Brown et al., "Language Models are Few-Shot Learners", 2020, Sec. 2.2 describes dataset mixture, filtering, and deduplication for GPT-3 pretraining.

Tokens: The Unit of Text

A token is an element of the model vocabulary. Text is converted to token IDs before entering the neural network.

Word-level problem

A fixed word dictionary cannot handle all misspellings, names, code, and punctuation well.

Character-level problem

Character sequences are long and force the model to relearn word structure.

Tokenization compromise

Use common words, word fragments, punctuation, spaces, and individual characters.

Bishop describes tokenization as a compromise between word-level and character-level representations.

References: C. M. Bishop and H. Bishop, *Deep Learning: Foundations and Concepts*, Springer, 2024, Sec. 12.2.2; T. Brown et al., "Language Models are Few-Shot Learners", 2020, Sec. 2.1.

Tokenization Caveats

Language

Different languages may need different numbers of tokens for the same idea.

Symbols

Mathematical notation, code, and rare names may be split into unintuitive pieces.

Context

The context window counts tokens, not characters or pages.

A tokenization problem can become a reasoning problem if important notation is chopped up badly.

Reference: C. M. Bishop and H. Bishop, *Deep Learning: Foundations and Concepts*, Springer, 2024, Sec. 12.2.2, for tokenization as a compromise between word-level and character-level representations.

Embeddings: IDs Become Vectors

After tokenization, a discrete ID is mapped to a dense vector by an embedding matrix.

If x_n is the one-hot vector for token n, the embedding vector is

$$v_n = Ex_n.$$

Because x_n is one-hot, this is equivalent to selecting one column of E.

Discrete input

Token IDs are symbols, not continuous numbers with natural distance.

Learned table

The columns of E are parameters learned from training data.

Context later

The same token embedding is later changed by Transformer layers using context.

Reference: C. M. Bishop and H. Bishop, *Deep Learning: Foundations and Concepts*, Springer, 2024, Sec. 12.2.1, Eq. 12.26, for word embeddings as matrix lookup.

Pretraining Creates Its Own Labels

Self-supervised language modeling turns ordinary text into many prediction tasks.

Token context	Training target
The	capital
The capital	of
The capital of	France
The capital of France	is

No human writes a label for each row. The next token already appears in the text.

Reference: T. Brown et al., "Language Models are Few-Shot Learners", 2020, Sec. 2.1, describes autoregressive Transformer language-model pretraining.

Pretraining Objective

For a GPT-style model, the sequence probability is decomposed left-to-right.

$$p_\theta(x_1, \ldots, x_N) = \prod_{n=1}^{N} p_\theta(x_n \mid x_1, \ldots, x_{n-1}).$$

Pretraining chooses parameters that assign high likelihood to observed training sequences:

$$\max_\theta \sum_{n=1}^{N} \log p_\theta(x_n \mid x_{<n}).$$

This objective teaches prediction of text continuations. Any facts, grammar, or reasoning patterns are learned only insofar as they help this prediction task.

References: C. M. Bishop and H. Bishop, *Deep Learning: Foundations and Concepts*, Springer, 2024, Sec. 12.2.4, Eq. 12.31; T. Brown et al., "Language Models are Few-Shot Learners", 2020, Sec. 2.1.

Loss Function: What Is Minimized?

In practice we minimize negative log-likelihood, usually implemented as cross-entropy loss.

For one token position:

$$\mathcal{L}_t(\theta) = -\log p_\theta(x_t \mid x_{<t}) = -\sum_{v \in V} y_v \log p_\theta(v \mid x_{<t}).$$

Vocabulary

All possible token IDs the model can choose from.

Target vector

One-hot target: 1 only for the correct token.

Model distribution

The model's predicted probability over the vocabulary.

Reference: C. M. Bishop and H. Bishop, *Deep Learning: Foundations and Concepts*, Springer, 2024, Chs. 5 and 12, for cross-entropy training and language-model objectives.

Loss Function: A Concrete Example

Context:

> The capital of France is

Suppose the correct next token is " Paris".

Good prediction

p(" Paris") = 0.80 gives loss about 0.22.

Bad prediction

p(" Paris") = 0.01 gives loss about 4.61.

Training pressure

The optimizer changes weights to make the correct token less surprising.

> So the immediate training goal is not "understand France"; it is "assign higher probability to the observed next token in context."

Reference: C. M. Bishop and H. Bishop, *Deep Learning: Foundations and Concepts*, Springer, 2024, Ch. 5 for cross-entropy / negative log-likelihood training.

What the Loss Pushes the Model To Do

Raise correct probability

If the real next token is "the", the loss decreases when the model assigns "the" more probability.

Use context

The model must use earlier tokens to predict which continuation is likely.

Average over scale

The final loss averages over many positions, documents, and batches.

This objective does not explicitly say "learn grammar" or "learn facts"; those abilities emerge because they help predict text.

The loss rewards probable continuation, not guaranteed truth. This is why verification is still needed.

Reference: T. Brown et al., "Language Models are Few-Shot Learners", 2020, for next-token language-model pretraining; reliability claim is a consequence of the objective, not a theorem about truth.

The Training Loop

A language model is a parameterized function. Training adjusts its weights to reduce average cross-entropy loss.

Step	Operation
1	sample a batch of token sequences
2	compute next-token probability distributions
3	compute cross-entropy against the real next tokens
4	backpropagate gradients through the Transformer
5	optimizer updates weights

This is statistical learning from text examples, not symbolic proof search.

Reference: C. M. Bishop and H. Bishop, *Deep Learning: Foundations and Concepts*, Springer, 2024, Chs. 5 and 12, for backpropagation, optimization, and Transformer training.

What Pretraining Gives You

Language

Grammar, style, translation patterns, summarization patterns.

Knowledge

Facts and associations reflected in the training distribution.

Code

Syntax, APIs, conventions, and common debugging patterns.

Reasoning patterns

Examples of explanations, proofs, calculations, and plans.

Transfer

The same model can be prompted for many tasks.

Weakness

Pretraining alone does not guarantee truth, safety, or instruction following.

References: T. Brown et al., "Language Models are Few-Shot Learners", 2020; L. Ouyang et al., "Training language models to follow instructions with human feedback", 2022.

Part III

Transformer Internals and LLM Training Stages

Transformer: The Core Problem

The same token embedding must become different contextual representations in different sentences.

Input

A sequence matrix X with N rows, one row per token position.

Attention stage

Mix information across rows: which other positions matter for this position?

MLP stage

Transform features inside each row using a shared feed-forward network.

Bishop's description is useful pedagogically: attention mixes token positions; the feed-forward network changes features within each token vector.

References: A. Vaswani et al., "Attention Is All You Need", 2017, Sec. 3; C. M. Bishop and H. Bishop, *Deep Learning: Foundations and Concepts*, 2024, Sec. 12.1.1.

Original Transformer Architecture

Encoder side

The original Transformer used stacked encoder layers for the input sequence.

Decoder side

The decoder used masked self-attention plus attention over encoder outputs.

LLM connection

GPT-style language models keep the left-to-right decoder idea and use causal masking.

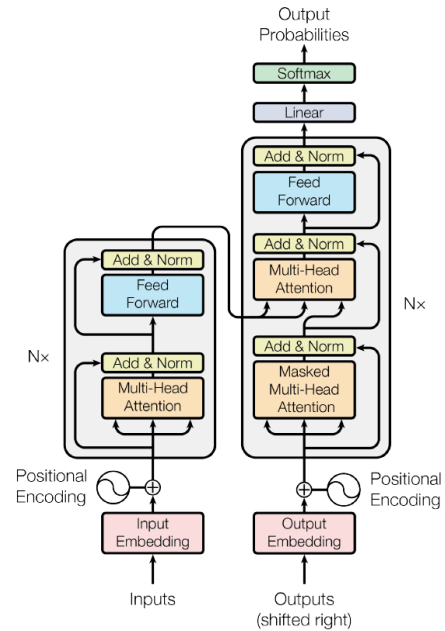


Figure 1: The Transformer - model architecture.

Figure from Vaswani et al. (2017).

Image/reference: A. Vaswani et al., "Attention Is All You Need", NeurIPS 2017, Fig. 1 and Sec. 3.

Sequence as a Matrix

After embedding, one passage of text is represented by a matrix X.

$$\boldsymbol{X} \in \mathbb{R}^{N \times D}$$

N = number of token positions, D = embedding / model dimension.

Each row x_n^T is the vector representation of one token position.

A Transformer layer maps X to another N x D matrix. It preserves the number of token positions while changing their representations.

Reference: C. M. Bishop and H. Bishop, *Deep Learning: Foundations and Concepts*, Springer, 2024, Sec. 12.1.1 and Fig. 12.3.

What One Transformer Layer Does

Mix positions

Self-attention lets one token use information from other tokens.

Transform features

A shared MLP applies a nonlinear transformation to each token vector.

Train deeply

Residual connections and layer normalization keep stacked layers stable.

Bishop & Bishop, Chapter 12, describes the same structural split: attention mixes the rows of the sequence matrix; the feed-forward network changes the features inside each row.

So a Transformer block is not only attention. It is attention plus MLP plus normalization/residual structure.

References: A. Vaswani et al., "Attention Is All You Need", 2017, Sec. 3.1; C. M. Bishop and H. Bishop, *Deep Learning: Foundations and Concepts*, 2024, Ch. 12.

Attention as a Weighted Sum

For output position n, attention forms a new vector by a weighted sum:

$$y_n = \sum_{m=1}^{N} a_{nm} x_m, \qquad a_{nm} \geq 0, \qquad \sum_{m} a_{nm} = 1.$$

Large weight

More information from token m flows into output n.

Small weight

Token m has little influence on output n.

Data-dependent

The weights are computed from the current input, not fixed by hand.

Reference: C. M. Bishop and H. Bishop, *Deep Learning: Foundations and Concepts*, Springer, 2024, Sec. 12.1.2, Eqs. 12.2-12.4.

Q/K/V Intuition From Recommendations

Bishop introduces Q/K/V using an information-retrieval analogy: search for a movie using a query and item keys.

Query

User profile or current request: what am I looking for?

Key

Movie/product/post descriptor: what does this item offer?

Value

The content returned or mixed in after matching.

Netflix, Amazon, or Douban are useful analogies, but Transformer Q/K/V are learned vectors inside the network.

References: C. M. Bishop and H. Bishop, *Deep Learning: Foundations and Concepts*, Springer, 2024, Sec. 12.1.3, for the movie-service query/key/value analogy; A. Vaswani et al., "Attention Is All You Need", 2017, Sec. 3.2.1.

Q/K/V Inside a Transformer

Starting from the sequence matrix X, the layer learns three projections:

$$Q = XW_Q, \qquad K = XW_K, \qquad V = XW_V.$$

QK^T

Every query position is compared with every key position.

softmax rows

Each row becomes attention weights that sum to one.

multiply by V

The weights combine value vectors into contextual outputs.

References: A. Vaswani et al., "Attention Is All You Need", 2017, Sec. 3.2.1; C. M. Bishop and H. Bishop, *Deep Learning: Foundations and Concepts*, 2024, Sec. 12.1.4, Eqs. 12.10-12.13.

Scaled Dot-Product Attention

The standard computation used in the Transformer paper is:

$$A = \mathrm{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right), \qquad Y = AV.$$

Scores

QK^T compares every query position with every key position.

Softmax

Each row becomes an attention distribution over positions.

Output

Multiplying by V forms one updated vector for each position.

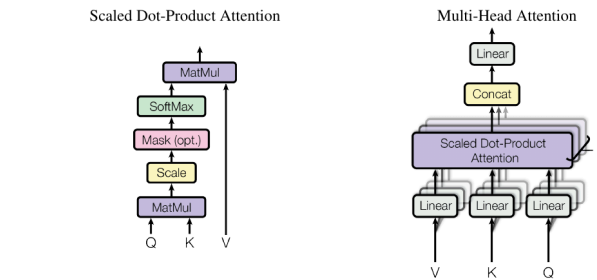


Figure 2: (left) Scaled Dot-Product Attention. (right) Multi-Head Attention consists of several attention layers running in parallel.

Figure from Vaswani et al. (2017).

Image/reference: A. Vaswani et al., "Attention Is All You Need", NeurIPS 2017, Fig. 2 and Sec. 3.2.1.

Why Divide by sqrt(d_k)?

Dot products grow

If vector components are roughly unit scale, q·k has variance proportional to d_k.

Softmax saturates

Very large positive or negative scores make probabilities close to 0 or 1.

Training suffers

Saturated softmax gives weak gradients and unstable learning.

The factor sqrt(d_k) keeps attention scores at a reasonable scale before softmax.

This is a numerical design choice that helps optimization; it is not a separate reasoning module.

Reference: A. Vaswani et al., "Attention Is All You Need", 2017, Sec. 3.2.1, explains the scaling by $\sqrt{d_k}$.

Multi-Head Attention

Several heads

Each head has its own learned Q, K, and V projections.

Parallel attention

Heads compute attention maps in parallel on the same sequence.

Concatenate + project

Head outputs are concatenated and linearly projected back to dimension D.

Do not overinterpret a head as a human-labeled module. Heads are learned components optimized by the training loss.

References: A. Vaswani et al., "Attention Is All You Need", 2017, Sec. 3.2.2 and Fig. 2; C. M. Bishop and H. Bishop, *Deep Learning: Foundations and Concepts*, 2024, Sec. 12.1.6.

The Full Transformer Block

Multi-head attention

Mixes information across token positions.

Residual + normalization

Adds the old representation back and stabilizes scale.

Position-wise MLP

The same feed-forward network transforms each row independently.

Vaswani et al. use sublayer output followed by residual connection and layer normalization. Bishop notes modern variants may use pre-norm, where normalization is applied before the sublayer.

References: A. Vaswani et al., "Attention Is All You Need", 2017, Sec. 3.1; C. M. Bishop and H. Bishop, *Deep Learning: Foundations and Concepts*, 2024, Sec. 12.1.7.

Positional Information

Problem

Self-attention by itself does not encode the order of token positions.

Language needs order

"not good" and "good not" contain similar tokens but different structure.

Solution

Add positional encodings or learned position embeddings to token embeddings.

Reference: A. Vaswani et al., "Attention Is All You Need", 2017, Sec. 3.5, positional encoding.

Transformer Families

Family	Attention pattern	Common use
Encoder	can attend bidirectionally within the input	representation, classification, retrieval
Decoder	causal self-attention: left-to-right generation	GPT-style language modeling
Encoder-decoder	decoder attends to encoder outputs	sequence-to-sequence translation/summarization

References: A. Vaswani et al., "Attention Is All You Need", 2017, Sec. 3, original encoder-decoder Transformer; T. Brown et al., "Language Models are Few-Shot Learners", 2020, GPT-style decoder-only language modeling.

GPT-Style Decoder: Training Target

A decoder language model represents a probability distribution for the next token:

$$p(x_1, \ldots, x_N) = \prod_{n=1}^{N} p(x_n \mid x_1, \ldots, x_{n-1}).$$

Input

Previous tokens in the context.

Output

A probability distribution over the vocabulary.

Loss

Cross-entropy penalizes low probability on the true next token.

During training the full sequence is known, but the model is masked so it cannot use future tokens to predict the present token.

Reference: T. Brown et al., "Language Models are Few-Shot Learners", 2020, Sec. 2.1, autoregressive language modeling.

Causal Mask

Autoregressive rule

Position n may use tokens 1,...,n, but not tokens to its right.

Training

The full sequence is known, but future positions are hidden from each prediction.

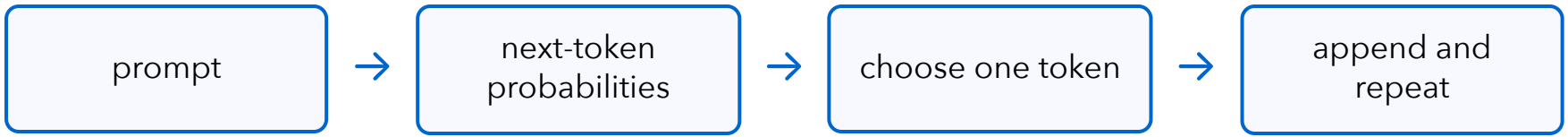
Implementation

Future attention scores are set to a very negative value before softmax.

For positions i and j in a decoder layer, the mask permits attention only when $j \leq i$.

References: A. Vaswani et al., "Attention Is All You Need", 2017, Sec. 3.2.3; T. Brown et al., "Language Models are Few-Shot Learners", 2020.

Inference: Text Is Generated One Token at a Time



This is autoregressive generation. The model repeatedly extends its own context.

Training uses next-token prediction over many text positions. Inference uses the trained model to repeatedly sample or choose the next token.

References: T. Brown et al., "Language Models are Few-Shot Learners", 2020; A. Holtzman et al., "The Curious Case of Neural Text Degeneration", 2019, for decoding behavior.

Decoding Choices

Greedy

Always choose the most likely next token. Deterministic, often repetitive.

Top-k

Sample only from the k most likely next tokens.

Top-p

Sample from the smallest set whose total probability reaches p.

Decoding is not training. It is the rule used at inference time to turn probabilities into actual text.

Temperature is important enough to discuss separately: it changes the probability distribution before sampling.

Reference: A. Holtzman et al., "The Curious Case of Neural Text Degeneration", 2019, discusses common decoding strategies including sampling constraints.

Temperature

Before choosing the next token, the model produces logits: one score for each token in the vocabulary.

Temperature rescales these logits before softmax:

$$p_T(v \mid c) = \frac{\exp(z_v/T)}{\sum_{u \in V} \exp(z_u/T)}.$$

T = 1

Use the model's ordinary softmax distribution.

0 < T < 1

Sharper distribution: high-probability tokens dominate more.

T > 1

Flatter distribution: lower-probability tokens get more chance.

Reference: A. Holtzman et al., "The Curious Case of Neural Text Degeneration", 2019, for sampling-based decoding context; formula here is the standard softmax temperature rescaling.

Temperature: Concrete Effect

Suppose the context is:

> The proof follows by

and the model's next-token logits favor three continuations.

Token	T = 0.5	T = 1	T = 2
induction	0.879	0.705	0.547
contradiction	0.119	0.260	0.331
compactness	0.002	0.035	0.122

> Temperature does not change the model's weights, knowledge, or reasoning ability. It only changes how the next token is sampled.

Reference: Temperature is an inference-time decoding parameter; see A. Holtzman et al., "The Curious Case of Neural Text Degeneration", 2019.

LLM Training Stages

Pretraining

Question: what continuation is statistically likely?

SFT

Question: how should an assistant answer an instruction?

RL / preference tuning

Question: which answer is preferred under a feedback signal?

These are different stages. Pretraining learns broad language and world patterns; post-training shapes behavior.

None of these stages is the same thing as formal verification.

References: T. Brown et al., "Language Models are Few-Shot Learners", 2020, for pretraining; L. Ouyang et al., "Training language models to follow instructions with human feedback", 2022, for SFT/RLHF.

From Base Model to Assistant

Base model

Pretrained mainly to predict the next token on large text corpora.

SFT

Supervised fine-tuning teaches the model to imitate high-quality instruction responses.

Preference tuning

RLHF-style methods use preferences to push outputs toward desired behavior.

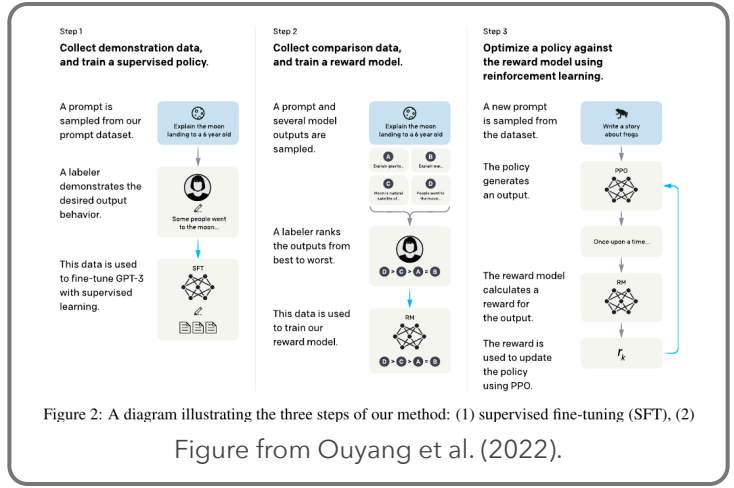


Figure 2: A diagram illustrating the three steps of our method: (1) supervised fine-tuning (SFT), (2)

Figure from Ouyang et al. (2022).

Image/reference: L. Ouyang et al., "Training language models to follow instructions with human feedback", 2022, Fig. 2 and Sec. 3, for SFT, reward modeling, and PPO-based RLHF.

Supervised Fine-Tuning

SFT trains on curated examples: instruction -> good response.

Data

Prompts, tasks, conversations, tool-use examples, and desired answers.

Objective

Usually still next-token loss, but on the desired assistant response.

Effect

Better instruction following, answer format, and task completion habits.

SFT is supervised learning: imitate demonstrations selected by humans or by a data pipeline.

SFT can improve usefulness, but it does not by itself guarantee truth or mathematical correctness.

Reference: L. Ouyang et al., "Training language models to follow instructions with human feedback", 2022, Sec. 3.2, supervised fine-tuning on demonstration data.

RL and Preference Tuning in LLMs

Preference data

Two or more model answers are compared; one is marked better.

Reward signal

A reward model or direct preference objective estimates preferred behavior.

Policy update

The model is adjusted to produce preferred outputs while staying close to the base policy.

RLHF often means: collect preferences, train a reward model, then optimize the language model against that reward.

Here "reinforcement learning" means optimizing behavior against a feedback signal. It does not mean the model has discovered mathematical truth.

References: L. Ouyang et al., "Training language models to follow instructions with human feedback", 2022, Secs. 3.3-3.4; R. Rafailov et al., "Direct Preference Optimization", 2023.

Keep These Concepts Separate

Transformer

The neural architecture: attention, MLP, residuals, normalization, position information.

Training

How weights are learned: pretraining, SFT, preference/RL methods.

Decoding

How text is generated at inference time: greedy, sampling, top-p, temperature.

This separation avoids a common confusion: changing temperature is not fine-tuning, and SFT/RL do not change the Transformer block formula.

References: A. Vaswani et al., "Attention Is All You Need", 2017, for architecture; L. Ouyang et al., InstructGPT/RLHF, 2022, for post-training; A. Holtzman et al., Neural Text Degeneration, 2019, for decoding.

Why Fluency Is Not Truth

A generated answer can be grammatical, confident, and false.

Objective

Pretraining rewards plausible continuations.

Training data

Contains errors, gaps, and mixed-quality sources.

Post-training

Improves assistant behavior, not mathematical certainty.

References: T. Brown et al., "Language Models are Few-Shot Learners", 2020; L. Ouyang et al., "Training language models to follow instructions with human feedback", 2022.

Sources: Transformer and Post-Training

Topic	Sources used
Transformer architecture	Vaswani et al., Attention Is All You Need (2017); Bishop and Bishop, *Deep Learning: Foundations and Concepts*, Ch. 12
Self-attention and multi-head attention	Vaswani et al. (2017), Sec. 3.2; Bishop and Bishop, Ch. 12.1
Next-token language modeling	Brown et al., Language Models are Few-Shot Learners (2020)
SFT and RLHF	Ouyang et al., Training language models to follow instructions with human feedback (2022)
Direct preference objectives	Rafailov et al., Direct Preference Optimization (2023)
Temperature and decoding	Holtzman et al., The Curious Case of Neural Text Degeneration (2019)

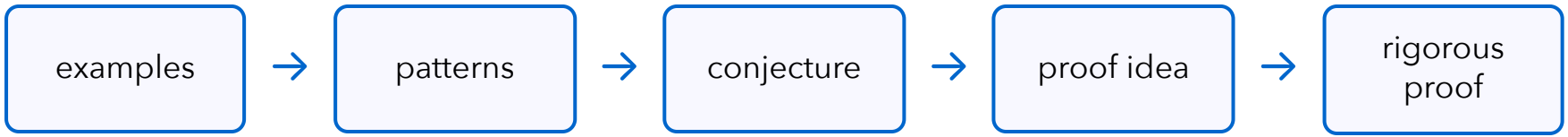
Mathematical Discovery and Course Projects

Examples, conjectures, proof critique, formalization, and verification

Part IV

How Mathematical Discovery Often Happens

A Clean Discovery Pattern



This is the clean version. Real discovery loops backward many times.

References: G. Pólya, *How to Solve It*, Princeton University Press; I. Lakatos, *Proofs and Refutations*, Cambridge University Press.

What Actually Happens

Compute

Make tables, test small cases, draw diagrams.

Compare

Look for similar behavior and exceptional cases.

Approximate

Find the right scale before the exact law.

Rephrase

Change notation or move to another domain.

Prove

Turn the pattern into necessary implications.

Refine

Fix assumptions and sharpen the statement.

References: G. Pólya, *How to Solve It*; I. Lakatos, *Proofs and Refutations*.

Project-Scale Discovery Tasks

Each item below can become a bounded experiment or tool workflow.

Examples

Generate or compute data.

Patterns

Suggest conjectures.

Reformulation

Find equivalent statements or useful language.

Proof critique

Find gaps and hidden assumptions.

Formalization

Translate into Lean.

Verification

Check claims with tools.

Part V

Prime Number Theorem: A Discovery Case Study

Scope of the PNT Case Study

You do not need analytic number theory today.

Data

Prime tables suggested a pattern.

Conjecture

The pattern became a precise asymptotic statement.

Reformulation

The proof required new analytic language.

We will track numerical evidence, conjectural formulas, and analytic reformulation. The proof itself is outside today's scope.

Reference: T. M. Apostol, "The Story of the Prime Number Theorem", *The Mathematical Intelligencer*, 1996.

The Question

Let π(x) be the number of primes less than or equal to x.

How fast does π(x) grow?

Before the theorem was proved, prime tables suggested a stable asymptotic pattern.

Gauss later reported noticing regularity in prime tables long before the theorem was proved.

References: T. M. Apostol, "The Story of the Prime Number Theorem", 1996; Encyclopaedia Britannica, "Prime number theorem".

Count the Primes

x	π(x)	x / log x
10	4	4.34
100	25	21.71
1,000	168	144.76
10,000	1,229	1,085.74
100,000	9,592	8,685.89

> The approximation is not exact, but it has the right order of growth.

References: T. M. Apostol, "The Story of the Prime Number Theorem", 1996; prime counts are standard values of $\pi(x)$ for $x=10,\ldots,100000$.

Gauss and Legendre

Gauss later wrote to Encke that his work with prime tables in 1792/1793 led him toward logarithmic estimates for prime counting.

Legendre published an approximation for prime counts in 1798 and refined it in later editions.

Accuracy point: they did not prove the Prime Number Theorem. They observed and formulated the asymptotic pattern.

The history starts from observation and approximation, not from a completed proof.

Reference: T. M. Apostol, "The Story of the Prime Number Theorem", 1996, discusses Gauss's letter to Encke and Legendre's 1798 approximation.

The Emerging Law

$$\pi(x) \sim x / \log x$$

This means that the ratio between π(x) and x / log x tends to 1 as x grows.

Reference: Encyclopaedia Britannica, "Prime number theorem", for the asymptotic statement $\pi(x)\sim x/\log x$.

Euler's Product

Euler connected primes to an analytic object:

$$\zeta(s) = \sum_{n=1}^{\infty} \frac{1}{n^s} = \prod_{p \text{ prime}} \frac{1}{1 - p^{-s}}.$$

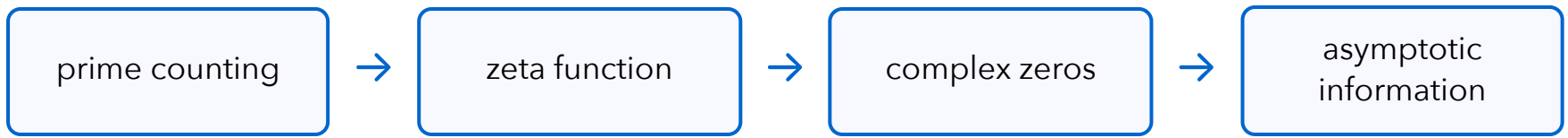
For this product formula, take Re(s) > 1.

The product says that information about primes is encoded in the zeta function.

Reference: T. M. Apostol, *Introduction to Analytic Number Theory*, Springer, 1976, Ch. 11; Euler product formula stated for $\operatorname{Re}(s)>1$.

Riemann's Reformulation

In 1859, Riemann linked prime counting to complex analysis and the zeros of $\zeta(s)$.



References: B. Riemann, "On the Number of Prime Numbers less than a Given Quantity", 1859, translated by D. R. Wilkins; T. M. Apostol, "The Story of the Prime Number Theorem", 1996.

Proof in 1896

Hadamard and de la Vallee Poussin independently proved the Prime Number Theorem in 1896.

data tables

approximate law

analytic reformulation

new proof machinery

theorem

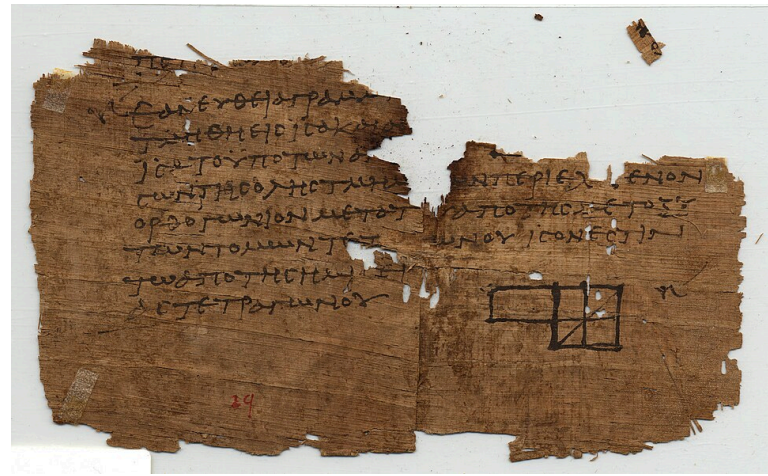
Reference: T. M. Apostol, "The Story of the Prime Number Theorem", 1996, for Hadamard and de la Vallee Poussin's independent 1896 proofs.

Project-Scale Tasks in the PNT Story

Do not make the project "prove PNT."

A project-scale version can focus on one artifact: tables, conjectures, reformulations, proof checks, or citation maps.

The object of study must be small enough to evaluate honestly.



The PNT story involves tables, formulas, analytic reformulation, and proof.

Reference for the PNT historical sequence: T. M. Apostol, "The Story of the Prime Number Theorem", 1996.

Sources: Prime Number Theorem Case

Claim area	Sources used
Gauss, Legendre, Riemann, Hadamard, de la Vallee Poussin	Apostol, The Story of the Prime Number Theorem
Riemann's 1859 memoir	Riemann, On the Number of Prime Numbers less than a Given Quantity
Prime number theorem statement	Encyclopaedia Britannica, Prime number theorem
Euler product formula	Apostol, *Introduction to Analytic Number Theory*, Ch. 11; standard statement for Re(s) > 1

Part VI

Agents, Harnesses, Tools, Memory, Skills

Why LLM Alone Is Not Enough

A one-shot answer has no reliable feedback loop.

No verifier

The model may sound confident but be wrong.

No search log

We cannot inspect failed attempts.

No tools

It may need computation, retrieval, or Lean.

References: S. Yao et al., "ReAct: Synergizing Reasoning and Acting in Language Models", 2022; Lean project, Theorem Proving in Lean 4.

Model vs Agent

Model

Maps context to output tokens.

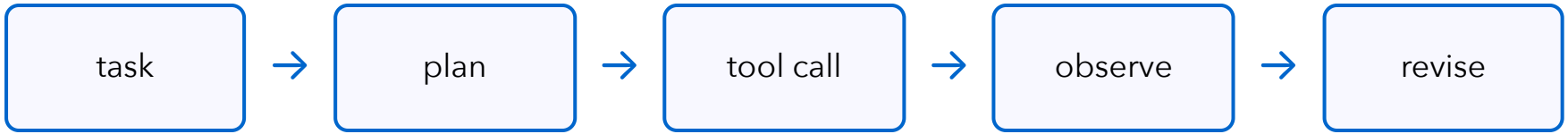
`prompt → completion`

Agent

A model inside a loop that can plan, call tools, observe results, revise, and stop.

References: S. Yao et al., "ReAct: Synergizing Reasoning and Acting in Language Models", 2022; Anthropic, Claude Code overview.

The Agent Loop



The observation can be a search result, a computation, a Lean error, or a human rubric.

Reference: S. Yao et al., "ReAct: Synergizing Reasoning and Acting in Language Models", 2022, for reasoning/action/observation loops.

What Is a Harness?

> The harness is the outer program that runs the model or agent.

Instructions

System prompt and policies.

Tools

What the agent can call.

Access

Files, network, and permissions.

Memory

Notes, logs, prior outputs.

Retries

When and how to try again.

Evaluation

How success is judged.

Reference: Anthropic, "Effective harnesses for long-running agents", engineering note, 2025; Anthropic, Claude Code SDK overview.

Tools for Math Agents

Search

Papers, web, theorem databases.

Python

Examples, data, counterexamples.

Lean

Formal proof checking.

CAS

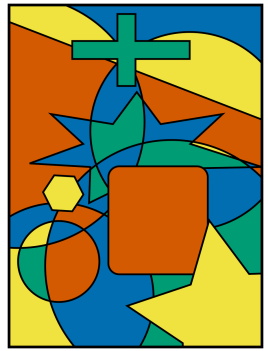
Symbolic algebra and simplification.

Retriever

Bring relevant facts into context.

Evaluator

Score outputs on a benchmark.

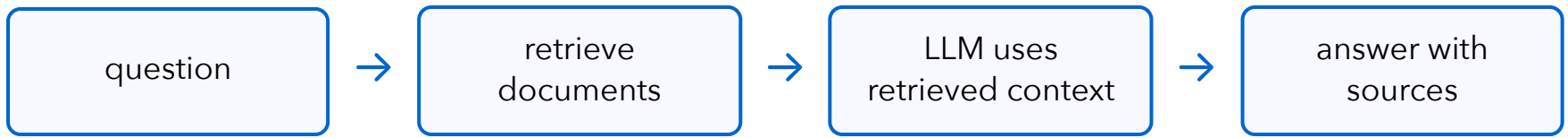


The four-color theorem is a familiar example where proof, computation, and checking meet.

References: Lean project, Theorem Proving in Lean 4; P. Lewis et al., "Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks", 2020, for retrieval-augmented generation.

Retrieval and RAG

Retrieval brings external information into the current context.



RAG means retrieval-augmented generation.

Reference: P. Lewis et al., "Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks", NeurIPS, 2020.

Memory, Skills, Benchmarks, Verifiers

Memory

Stored notes or logs across steps.

Skill

Reusable packaged procedure for a recurring task.

Benchmark

Fixed task set for evaluation.

Verifier

Independent checker such as Lean or a rubric.

Trace

Record of attempts, tool calls, and failures.

Metric

Numerical or structured success criterion.

References: S. Yao et al., "ReAct", 2022, for tool-use loops; Anthropic, "Effective harnesses for long-running agents", 2025, for harness structure.

Example Skill: Proof Review

> Input: an informal proof draft.

- Check definitions and quantifiers.
- Identify hidden assumptions.
- Test each implication in order.
- Look for counterexamples to vague claims.
- Separate critical errors from repairable gaps.
- Suggest formal statements that could verify key lemmas.

References: Lean project, Theorem Proving in Lean 4, for machine-checkable proof context; H. Lightman et al., "Let's Verify Step by Step", 2023, for step-level evaluation.

Part VII

Course Project Directions

Project Formats

Benchmark study

Benchmark and analysis: prompts, examples, metrics, error taxonomy.

Tool workflow

Small tool workflow: retrieval, checker, or retry loop.

Agent prototype

Agent prototype: planning, tools, logs, evaluation, and failure analysis.

The format is flexible. The key requirement is a clear question, a reproducible setup, and honest evaluation.

What Makes a Good Project?

A good project is small, measurable, and reproducible.

Question

One sentence, testable.

Dataset

Concrete examples or problems.

Baseline

A simple comparison method.

Metric

How success and failure are measured.

Artifact

Code, prompts, benchmark, report, demo.

Failure analysis

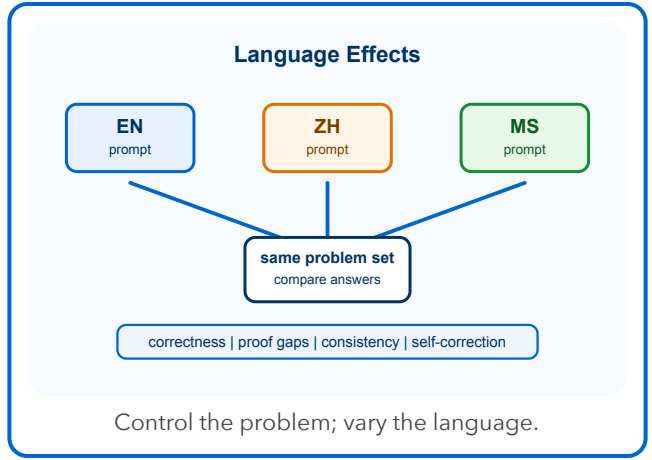
What goes wrong, and why?

Project 1: Language Effects

Question: Does prompt or solution language affect mathematical reasoning quality?

- Compare English, Chinese, Malay, and bilingual prompts.
- Use the same problem set and same evaluation rubric.
- Separate translation errors from reasoning errors.

Metrics: correctness, proof gaps, consistency across runs, self-correction rate.



Control the problem; vary the language.

Reference context: W. Luo, W. X. Zhao, J. Sha, S. Wang, J.-R. Wen, "MMATH: A Multilingual Benchmark for Mathematical Reasoning", arXiv:2505.19126, submitted May 25, 2025.

Project 2: Prover Agent Design

Question: Can an agent improve proof generation by planning, checking, and revising?

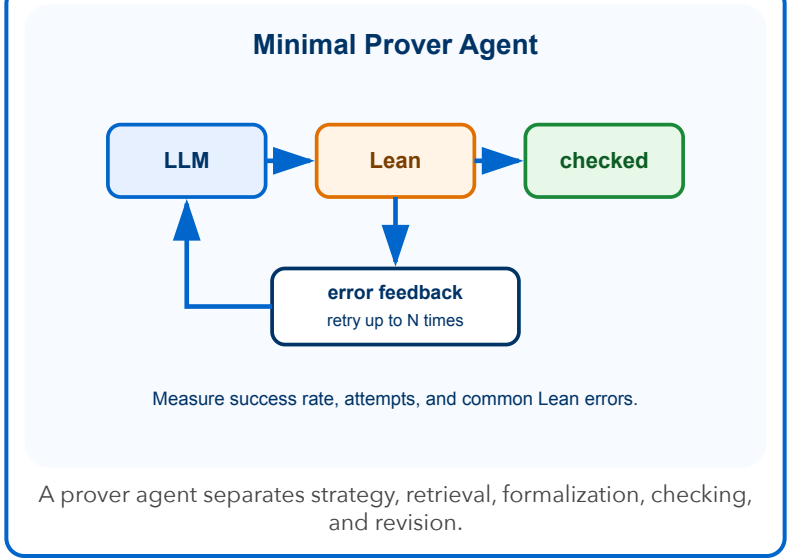
informal proof strategy

lemma retrieval or theorem search

Lean statement or proof skeleton

verification error and revision log

Metrics: proof success rate, useful lemma rate, number of revisions, human repair time.



A prover agent separates strategy, retrieval, formalization, checking, and revision.

References: S. Yao et al., "ReAct: Synergizing Reasoning and Acting in Language Models", 2022, for agent loops; Lean project, Theorem Proving in Lean 4, for formal checking context.

Rethlas / Archon as Inspiration

This is inspiration for the prover-agent project, not a requirement to reproduce a research system.

Rethlas

Informal reasoning agent for exploring strategies.

Matlas

Theorem search used by Rethlas.

Archon

Formal verification agent that builds Lean 4 projects.

Possible metrics: useful lemma rate, skeleton quality, type-checking rate, human repair time.

Reference: H. Ju et al., "Automated Conjecture Resolution with Formal Verification", arXiv:2604.03789, submitted April 4, 2026; abstract defines Rethlas, Matlas, Archon, and LeanSearch roles.

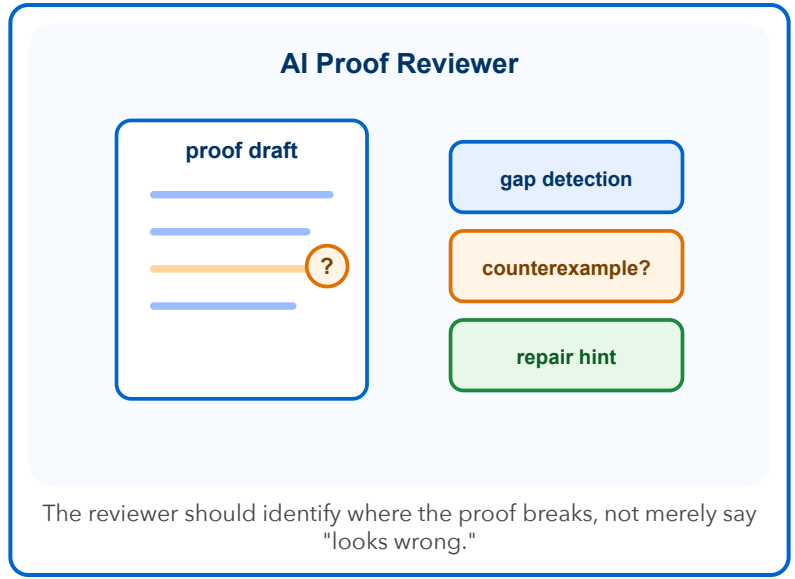
Project 3: AI Proof Reviewer

Question: Can an LLM find gaps in informal mathematical proofs?

Dataset

- Correct proofs.
- Subtly flawed proofs with planted errors.
- Incomplete proofs with missing justifications.

Metrics: precision/recall on planted errors, repair suggestion quality.



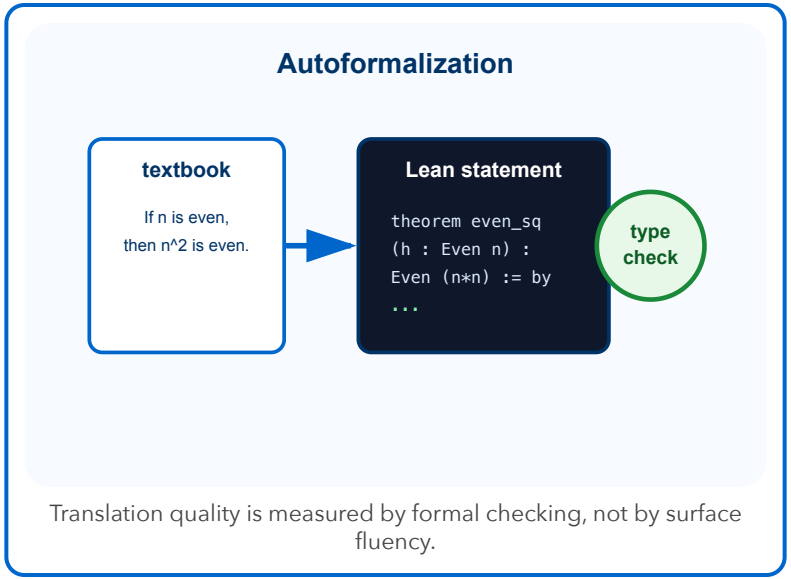
The reviewer should identify where the proof breaks, not merely say "looks wrong."

Reference context: H. Lightman et al., "Let's Verify Step by Step", arXiv:2305.20050, 2023, for step-level evaluation of mathematical reasoning; Lean docs for machine-checkable proof context.

Project 4: Autoformalization

Question: How well can LLMs translate informal statements into Lean statements or proof skeletons?

- Start with elementary algebra, number theory, or real analysis statements.
- Record whether the generated Lean type-checks.
- Classify failures: wrong variables, missing assumptions, wrong library names.



Translation quality is measured by formal checking, not by surface fluency.

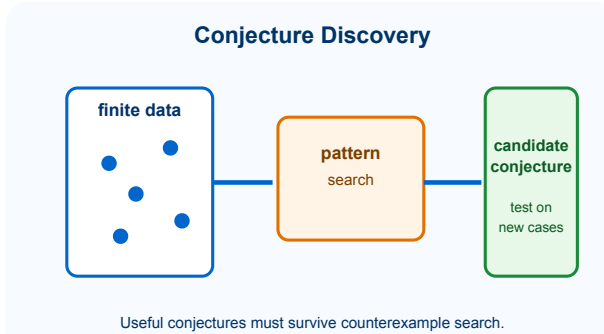
Reference: Y. Wu, A. Q. Jiang, W. Li, M. N. Rabe, C. Staats, M. Jamnik, C. Szegedy, "Autoformalization with Large Language Models", arXiv:2205.12615, 2022.

Project 5: Automated Conjecture Discovery

Question: Can simple ML or symbolic search propose plausible mathematical conjectures?

- Possible domains: integer sequences, graph invariants, finite groups, polynomial identities.
- Generate conjectures from small examples.
- Use held-out cases and counterexample search before trusting a pattern.

Metrics: novelty, truth rate on held-out examples, proofability, counterexample rate.



A good project separates pattern finding from verification.

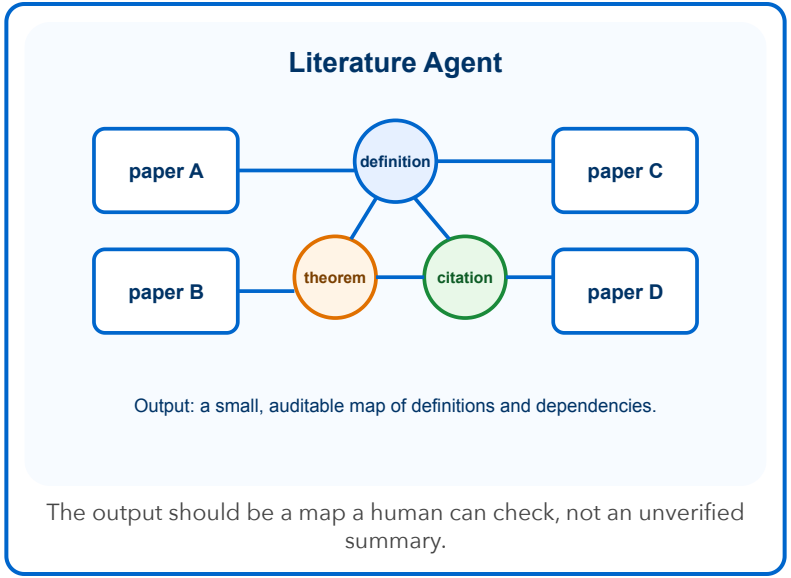
References: D. B. Lenat, "AM, an Artificial Intelligence Approach to Discovery in Mathematics as Heuristic Search", Stanford Ph.D. thesis, 1976; S. Colton, *Automated Theory Formation in Pure Mathematics*, Springer, 2002.

Project 6: Literature Agent

Question: Can an agent build a reliable map of a small mathematical topic?

- Choose a narrow topic: one theorem family or one technique.
- Extract definitions, assumptions, theorem dependencies, and citations.
- Audit every citation against the original source.

Metrics: citation accuracy, missing dependency rate, human correction time.



The output should be a map a human can check, not an unverified summary.

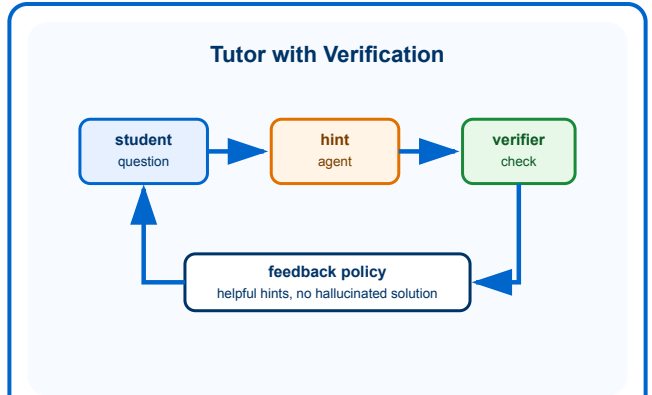
Reference context: P. Lewis et al., "Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks", NeurIPS 2020, for retrieval-augmented generation; project evaluation should audit cited original sources directly.

Project 7: Tutor with Verification

Question: Can a tutor agent teach problem solving while preventing hallucinated answers?

- **Hint first:** no direct full solution immediately.
- **Check claims:** use a checklist or Lean when possible.
- **Evaluate:** correctness and student usability.

Metrics: correct hints, avoided false claims, student success after hints.



A useful tutor is helpful and constrained by verification.

Reference context: H. Lightman et al., "Let's Verify Step by Step", 2023, for step-level feedback; Lean project, Theorem Proving in Lean 4, for formal-verification examples.

Feasibility Test

- Can the first dataset be built quickly?
- Can the baseline be run without a large system?
- Can success be measured without relying only on opinion?
- Would a negative result still be meaningful?
- Can the scope be reduced if the toolchain becomes difficult?

Project Milestones

Time	Milestone
Launch	project seed chosen
Checkpoint 1	dataset/problem set and baseline ready
Checkpoint 2	first prototype or evaluation table
Checkpoint 3	failure analysis and revised method
Clinic	project clinic and presentation rehearsal
Presentations	final presentation and artifact review

What Counts as Success?

> The project does not need to solve all of AI mathematics.

Clear experiment

The question is measurable.

Honest result

Positive or negative, the evidence is clear.

Useful artifact

Others can inspect prompts, data, logs, or code.