Claude & AI: The Complete Guide

September 2026 · Carter Luense / Lund Studio

What AI is, exactly how it works from the math up, how Claude was built, and which model to use.

Part 1: What Is AI

Artificial intelligence is software that can do things we used to think required a human brain — reading, writing, reasoning, coding, analyzing data, holding a conversation, making decisions.

The approach behind Claude, ChatGPT, and every conversational AI is a large language model (LLM) — a program that has read a massive amount of human writing and learned the patterns in it. Not memorized word-for-word, but absorbed the structure and logic the way a person absorbs grammar without reciting the rules.

Part 2: Exactly How It Works

Not a metaphor. Every piece of the machine, from the math up.

Step 1: Everything Becomes Numbers

Tokenization

The model sees "tokens" — chunks of text a tokenizer learned to treat as units. "The" is one token. "Unbelievable" might become three: "un" + "believ" + "able." Claude's vocabulary is ~100,000+ tokens. Every message is split into token IDs — just numbers — before the model touches it.

Embeddings

Each token ID becomes a dense vector of thousands of numbers — coordinates in a very high-dimensional space. "King" is a point. "Queen" is nearby. "Banana" is far away. These positions are learned during training: the model discovers that words with similar meanings should have similar vectors, that the direction from "man" to "woman" should roughly equal the direction from "king" to "queen." In a frontier model, each embedding is 8,000–16,000+ numbers.

Positional Encoding

Embeddings alone do not carry word order. "Dog bites man" and "man bites dog" would look identical. Positional encoding adds a mathematical signal based on each token's position. Modern models use Rotary Position Embeddings (RoPE), which rotate the vector by its position — letting the model understand both absolute position (this is token 47) and relative distance (these two tokens are 12 apart).

Step 2: The Transformer — Layer by Layer

The token sequence flows through a stack of identical layers — 80 to 120+ in a frontier model. Each layer has two components: attention and the feed-forward block.

Attention — The Core Mechanism

Every token creates three vectors from learned weight matrices: Query (what am I looking for?), Key (what do I contain?), Value (what information do I carry?). The model computes the dot product of every Query against every Key — measuring similarity. These scores are divided by √d (square root of the Key dimension) to prevent saturation, then passed through softmax to become probabilities summing to 1. Each token's output is a weighted sum of all Value vectors, weighted by those attention probabilities.

The equation: $\text{Attention}(Q, K, V) = \text{softmax}(QK^T / \sqrt{d_k}) \times V$

Multi-head attention: This runs 64–128+ times in parallel per layer, each head with different projection matrices, learning different relationship types — grammar in one head, co-reference in another, semantics in a third. Outputs are concatenated and projected back down.

Causal masking: During generation, each token can only see tokens before it, never after (they haven't been generated yet). Future positions are masked to negative infinity before softmax, zeroing them out. This makes the model autoregressive — it generates left to right.

Feed-Forward Block

After attention, each token passes independently through two large matrix multiplications with a nonlinear activation (SiLU or GELU) between them, through a space often 4× wider than the embedding dimension. Research suggests FFN layers store factual knowledge while attention layers route information.

Residual Connections and Layer Norm

The input to each sub-block is added back to its output (residual connection — preserves the signal). Before each sub-block, activations are normalized (RMSNorm — keeps magnitudes stable across 100+ layers). Each layer: normalize → attention → add residual → normalize → FFN → add residual. Repeat 80–120 times.

Step 3: Generating Output

The final layer's output for the last token is multiplied by the unembedding matrix, producing ~100,000 logits (one score per vocabulary token). These are divided by temperature (lower = more deterministic, higher = more creative), passed through softmax to become probabilities, then sampled — randomly selecting a token weighted by the probabilities. Top-k (only consider the k most likely tokens) and top-p / nucleus sampling (only consider tokens whose cumulative probability reaches p) shape the sample. The selected token is appended. The full model runs again for the next token. Repeat until a stop token.

Every word Claude writes = one complete forward pass through the entire network. A 1,000-word response is ~1,300 passes through hundreds of billions of parameters.

Step 4: How It Learns

Pre-Training

The model sees text with the next token hidden. It predicts a probability distribution. Cross-entropy loss measures how wrong it was. Backpropagation computes a gradient for every parameter (hundreds of billions of them) — "if I nudge this number, does the loss go up or down?" — using the chain rule of calculus through every operation in the network. The optimizer (AdamW) adjusts each parameter slightly in the direction that reduces loss, with per-parameter adaptive learning rates and weight decay. This runs trillions of times across hundreds of billions of words, on thousands of GPUs for months. A single training run costs $50–100+ million.

What emerges: the task is trivially simple (predict the next word). But to do it well, the model must learn math, programming, law, science, chess, medicine, and more. No one teaches it these things. They emerge from prediction pressure across enough data. A model trained on 10× more data doesn't just get 10× better — it develops qualitatively new capabilities. This is called emergence.

Fine-Tuning: RLHF + Constitutional AI

The raw model can complete any text but doesn't know how to be helpful. Supervised fine-tuning (SFT) trains it on curated conversation examples. RLHF then has it generate multiple responses, human raters rank them, a reward model learns what humans prefer, and the main model is optimized with PPO (Proximal Policy Optimization) to maximize reward while staying close to its base behavior.

Constitutional AI is Anthropic's innovation: instead of relying only on human raters, the model is given a written constitution — explicit principles for how to behave. It generates responses, critiques them against the constitution, revises them, and trains on the revised versions. A preference model is then trained on AI-generated comparisons (not human labels), and the base model is fine-tuned against it. This produces what Anthropic calls a "Pareto improvement" — the model becomes both more helpful AND more harmless, with zero human labels on harmlessness. All safety gains come from AI supervision alone.

Red-Teaming

Before release, adversarial teams try to break the model. It is further trained to resist these attacks. Anthropic publishes system cards documenting known capabilities, limitations, and risk profiles.

Step 5: Inference Engineering

KV-Cache: Stores previously computed Key and Value vectors so they aren't recomputed on every token. Turns per-token attention from O(n) to O(1). Without this, generation would be unusably slow.

Prompt caching: When requests share the same prefix (system prompt), the KV-cache for that prefix is precomputed and reused. Cache reads cost a fraction of full input ($0.25/M vs $10/M on Fable 5.1) — a 40× reduction.

Hardware: NVIDIA H100 and B200 GPUs (and Google TPUs, Amazon Trainium). The model is split across GPUs using tensor parallelism (splitting matrix multiplications) and pipeline parallelism (different layers on different GPUs). Multiple requests are batched. Continuous batching lets new requests enter as slots open.

Adaptive thinking: Newer models generate internal reasoning tokens the user doesn't see. Higher effort = more thinking = more cost = better answers on hard problems.

Tool use: The system prompt includes JSON descriptions of available tools. When the model determines a tool is needed, it generates a JSON function call instead of text. The system executes it, feeds the result back, and the model incorporates it.

Part 3: How They Made Claude

This is the story of how Claude went from an idea to the model you are talking to right now.

The Break: 2021

In 2021, Dario Amodei was the VP of Research at OpenAI — the company behind ChatGPT. His sister Daniela Amodei was VP of Operations. They were among the most senior people at what was then the leading AI lab in the world.

They left. They took several of OpenAI's top researchers with them. The reason was a fundamental disagreement about how carefully frontier AI should be developed and commercialized. Dario and Daniela believed that as models got more powerful, safety had to be at the center of the work, not bolted on afterward. They believed OpenAI was moving too fast and not taking the risks seriously enough.

They founded Anthropic in San Francisco with a mission statement that sounds simple but defines everything the company does: build AI systems that are reliable, interpretable, and steerable. In practice, this meant investing in safety research at the same level as capability research — not as a checkbox, but as the core of the product.

The Invention: Constitutional AI (2022)

Before Claude existed as a product, Anthropic published the research that would define it.

The standard method for making AI models behave well was RLHF — reinforcement learning from human feedback. You hire people to rate the model's responses, and the model learns to produce what the raters prefer. The problem: this is expensive, slow, inconsistent (different raters have different values), and you can never fully specify what "good behavior" means through examples alone.

Anthropic's answer was Constitutional AI. Instead of relying entirely on human raters, they wrote a document — a constitution — containing explicit principles for how the model should behave. The first version, published in 2022, had 75 guidelines. The model was trained to evaluate its own responses against these principles, critique itself, revise its answers, and learn from the revised versions.

The key result: the model became both more helpful and more harmless simultaneously, with zero human labels on the harmlessness side. All the safety improvement came from AI supervision against the written principles. This was a genuine breakthrough — it meant safety could scale with capability instead of requiring ever-larger teams of human raters.

The constitution is not a technical detail. It IS the value system. Every decision Claude makes about whether to help, refuse, warn, hedge, or be direct traces back to those principles. When Anthropic updates the constitution, Claude's behavior changes. The principles are published publicly so anyone can read and critique them.

Claude 1: The First Model (March 2023)

Anthropic launched the first Claude on March 14, 2023. It was invite-only. The context window was 9,000 tokens — tiny by today's standards. It competed with GPT-4, which had just launched days earlier. Claude immediately attracted enterprise customers who cared about safety and brand risk.

In May 2023, Claude became the first frontier model to reach a 100,000-token context window — a 10× increase that let users feed entire books and codebases into a single conversation. This was a signal of where Anthropic was headed: longer context, more capable, safety built in.

Claude 2: Going Public (July 2023)

Claude 2 launched July 11, 2023. It was the first version available to the general public through claude.ai. The context window grew to 200,000 tokens. Hallucination rates dropped. API access opened. Anthropic had a real product.

Claude 3: Becoming the Frontier (March 2024)

This was the release that changed Anthropic's position in the industry. Claude 3 came as a family of three models — Haiku (fast and cheap), Sonnet (balanced), and Opus (most capable) — a tiered structure that every subsequent release has followed.

Claude 3 Opus matched or exceeded GPT-4 on most benchmarks. It had a 200K-token context window, strong reasoning, nuanced instruction-following, and a personality users described as "thoughtful and careful." Anthropic was no longer just the safety-focused alternative. It was the frontier.

Claude 3 also introduced vision — the ability to understand images, not just text.

Claude 3.5 Sonnet: The Coding Breakout (Mid-2024)

Claude 3.5 Sonnet became one of the most widely used models for real-world software engineering. Developers praised its code generation, debugging, and refactoring. It powered Claude Code (Anthropic's terminal-based coding tool) and established Claude as the model developers reached for first. This was the moment Claude went from "strong contender" to "default choice" for a significant segment of the market.

Claude 3.5 Sonnet also introduced Computer Use (October 2024)

For the first time, an AI model could operate a computer — moving the mouse, clicking buttons, typing text, navigating applications. It was released as a public beta. The model could see the screen (as an image), decide what to do, and execute actions through simulated keyboard and mouse input. This was a category-defining moment.

Claude 4: Agents and Extended Thinking (May 2025)

Claude 4 moved deep into coding, agents, and advanced reasoning. Claude Opus 4 and Sonnet 4 launched together. Extended thinking (chain of thought) became a core feature — the model could reason internally before answering, dramatically improving performance on hard problems.

Opus 4 was the first Claude model to activate ASL-3 protections — Anthropic's highest deployed safety level, including real-time constitutional classifiers, restricted network traffic, external red-teaming, and bug bounties.

Claude 4.5 through 4.8: Rapid Iteration (Late 2025 – May 2026)

Anthropic shifted to rapid point releases within the Opus tier, each one refining the model for real work:

- **4.5 (November 2025):** Spreadsheets, long-running chats, high-intelligence coding. The 1M-token context window arrived.
- **4.6 (February 2026):** Code review, debugging, robust planning, trustworthy long-running agents. This is the model you are talking to right now.
- **4.7 (April 2026):** Improved agentic coding and reasoning, but drew criticism for being preachy and second-guessing users.
- **4.8 (May 2026):** Fixed 4.7's behavioral issues. Better honesty, restored creativity. Adaptive thinking on by default. The best of the 4.x line.

Claude 5 and the Mythos Tier (June–September 2026)

The current generation:

- **Fable 5 (June 9, 2026):** Introduced the Mythos tier — a new class above Opus. The most capable model Anthropic had ever released. Briefly pulled June 12–30 during a U.S. export-control order.
- **Sonnet 5 (June 30, 2026):** Near-Opus intelligence at Sonnet cost. Default on Free and Pro plans.
- **Opus 5 (July 24, 2026):** The recommended starting point for hard work. Half Fable's price. Solved all 6 IMO 2026 math problems. 4× better than any other model on abstract reasoning.
- **Fable 5.1 (September 1, 2026):** Current flagship. More than doubled Fable 5 on scientific research. 4× cheaper caching. Strongest model available.

The Infrastructure

Building Claude requires hardware and money at a scale most people cannot imagine:

- **Compute:** Thousands of NVIDIA H100/B200 GPUs, Google TPUs, and Amazon Trainium chips running in parallel. Data parallelism (splitting training data across chips), model parallelism (splitting the model itself across chips), and pipeline parallelism (different layers on different devices).
- **Training cost:** A single frontier training run costs $50–100+ million in compute alone. 10^{25} to 10^{26} floating-point operations.
- **Training data:** Hundreds of billions of words of text — books, websites, code, papers, conversations. No single person could read it in a thousand lifetimes.
- **Team:** Hundreds of researchers, engineers, and alignment scientists. The company has raised billions in funding from Google and Amazon.
- **Iteration cycle:** As of 2026, Anthropic is shipping major model updates every 1–2 months — faster than any previous stretch in Claude's history.

The Constitution

Everything Claude does traces back to a single document. Anthropic's constitution defines what Claude should value, how it should handle conflicts, when it should help, when it should refuse, how it should

be honest about uncertainty, and how it should treat people. The constitution is published publicly. It has been updated multiple times and was the subject of a public input experiment where ordinary people collectively shaped its principles.

This is what makes Claude Claude. Not the model weights. Not the benchmark scores. The constitution is why Claude tells you when it doesn't know something instead of making it up. Why it pushes back when you're wrong but doesn't lecture you. Why it refuses genuinely harmful requests without being annoyingly cautious about everything else. Other labs train their models with implicit values embedded in human ratings. Anthropic writes its values down, publishes them, and trains the model to follow them explicitly. That is the difference.

Part 4: The Proof

Task	Result	Score
Math olympiad	All 6 IMO 2026 problems — gold medal	42/42
Bug fixing	96% of real production bugs	SWE-bench 96.0%
Scientific research	Multi-step research workflows	T-B-Sci 52.6%
Novel reasoning	4× better than next-best model	ARC-AGI-3 30.2%
Computer use	Autonomous computer operation	OSWorld 77.9%
Knowledge work	Expert-level finance/legal/docs	GDPval 1,861
Agentic coding	Multi-hour autonomous sessions	Frontier-Bench 43.3%

Standardized tests. Independent organizations. Third-party verification. The model performs at or above skilled human professionals across disciplines, and it is improving by step changes every few months.

How Would You Build One Yourself

If you wanted to build your own LLM — not use someone else's, but actually build one from nothing — here is the real path, from the simplest version that costs $15 to the frontier that costs $100 million.

The Simplest Version: A Toy Model on Your Laptop

You can build and train a working language model on your own computer in a weekend. It will not be good. It will generate semi-coherent text at best. But it will work, and you will understand how every piece fits together.

What You Need

- **A computer with a GPU.** Any modern NVIDIA GPU with at least 8GB of VRAM works. An NVIDIA RTX 3060, 4060, or higher. A MacBook with Apple Silicon (M1/M2/M3/M4) can also work using Metal acceleration. No GPU at all? You can rent one in the cloud for $1–2/hour.
- **Python.** The language nearly all ML work is done in.
- **PyTorch.** The open-source framework that handles the math. It defines the neural network layers, runs the forward pass, computes the loss, runs backpropagation, and updates the parameters. pip install torch. Free.
- **A tokenizer.** Byte-Pair Encoding (BPE) is the standard. The HuggingFace tokenizers library lets you train one on your own text in a few lines of code. Or use a pre-trained tokenizer from an existing model.
- **Training data.** For a toy model, a few hundred megabytes of text works. Wikipedia dumps, Project Gutenberg books, or your own corpus. The more data, the better the model, but you can start with anything.

The Steps

1. Train or load a tokenizer — this converts raw text into token IDs. BPE learns the most efficient vocabulary from your corpus.

2. Define the model architecture — in PyTorch, you write a class that stacks: an embedding layer (token IDs → vectors), positional encoding (add position information), N Transformer blocks (each containing multi-head attention + feed-forward + layer norm + residuals), and an output head (vectors → vocabulary logits). A 15-million parameter model uses about 6 layers, 6 attention heads, and 384-dimensional embeddings. A 125-million parameter model (GPT-2 small equivalent) uses 12 layers, 12 heads, 768-dimensional embeddings.

3. Write the training loop — load batches of tokenized text, run the forward pass, compute cross-entropy loss against the actual next token, call loss.backward() (backpropagation), call optimizer.step() (parameter update). Repeat millions of times.

4. Train — on a single GPU, a 15M-parameter model trains on a small corpus in hours. A 125M-parameter model takes a day or two. You will see the loss drop from ~11 (random) to ~3–4 (coherent text).

5. Generate text — feed a prompt, sample the next token from the output distribution, append it, repeat. Adjust temperature, top-k, and top-p to control quality.

Cost: $0 if you own a GPU. $15–50 in cloud GPU rentals if you don't. The model will generate grammatical text and show basic pattern understanding, but it will not be useful for real work.

The Shortcut: Nanochat

There is an open-source project called nanochat that packages the entire pipeline — tokenization, pre-training, fine-tuning, evaluation, inference, and a chat UI — into a minimal codebase you can run on a single GPU node. You can train a GPT-2-capability model (which cost $43,000 in 2019) for about $48–72 in 2–3 hours on 8 rented H100 GPUs. On a spot instance, as low as $15–20.

The Serious Version: A Useful Small Model

To build a model that actually does useful work — answers questions in a specific domain, writes code, summarizes documents — the scale jumps:

Option A: Fine-Tune an Existing Open Model

This is what most people should do. Take an existing open-weights model (Llama, Mistral, Qwen) and fine-tune it on your own data:

- **Pick a base model.** Llama 3.1 8B or Mistral 7B are the standard starting points. They are already pre-trained on trillions of tokens. You are adapting them, not building from zero.
- **Prepare your data.** Hundreds to thousands of examples in the format you want (instruction + ideal response). Quality matters far more than quantity.
- **Fine-tune with LoRA/QLoRA.** These techniques update only a small fraction of the model's parameters (1–4%), which means you can fine-tune a 7B-parameter model on a single 16GB GPU. A fine-tuning run costs $2–25 in cloud GPU time and takes 1–4 hours.
- **Evaluate and iterate.** Test against benchmarks. Use a larger model as a judge. Fix failures in your training data and retrain.

Cost: $10–100 per fine-tuning run. A few hundred to a few thousand dollars total including experimentation. This is how most companies build custom AI in 2026.

Option B: Train From Scratch

If you need a model that is fundamentally different from anything that exists — a new language, a proprietary domain, a novel architecture — you train from scratch:

- **1–7 billion parameters:** $100K–$1.5M. Needs 8–64 GPUs running for days to weeks. Requires tens to hundreds of billions of tokens of training data. You need ML engineers who know distributed training.
- **8–70 billion parameters:** $1M–$10M+. Hundreds of GPUs. Months of training. A team of ML engineers, data engineers, and infrastructure specialists.
- **Frontier scale (hundreds of billions):** $50M–$100M+ just for compute. Thousands of GPUs. Months of training. A world-class team. This is what Anthropic, OpenAI, and Google do. There are fewer than ten organizations in the world that can do this.

The Full Stack: What You Actually Need

Whether you are building a toy or a frontier model, the same components are always present:

Component	What It Is	Tools
Tokenizer	Converts text to token IDs	HuggingFace tokenizers, SentencePiece, tiktoken
Model architecture	The Transformer itself	PyTorch, JAX, or Rust (hermes-llm)
Training data	The text the model learns from	Common Crawl, Wikipedia, books, code (The Stack), your own corpus
Training framework	Handles distributed training, mixed precision, checkpointing	PyTorch + DeepSpeed, Megatron-LM, Axolotl, nanochat
Hardware	GPUs that do the math	NVIDIA H100/B200/A100, Google TPU, Apple Silicon (small models)
Evaluation	Tests whether the model works	lm-eval-harness, custom benchmarks, LLM-as-judge
Inference engine	Serves the model to users	vLLM, TensorRT-LLM, llama.cpp (local), Ollama
Fine-tuning	Adapts the base model	LoRA/QLoRA via PEFT, Axolotl, Unsloth

The Honest Summary

Building a toy model: a weekend and $15. Educational. You will understand how LLMs work at a level most people never reach.

Fine-tuning an existing model for real use: $10–$1,000 and a few days. This is what most companies and developers should do.

Training a useful model from scratch: $100K–$10M, a team of specialists, and months of work.

Building a frontier model like Claude: $50–100M+ in compute alone, billions in total investment, hundreds of researchers, years of accumulated knowledge, and a published constitution that defines the model's values. There are fewer than ten organizations on Earth that can do this.

The barrier to entry is collapsing at the bottom. Anyone with a laptop and curiosity can build a toy model today. The barrier at the top is rising — the gap between a good small model and a frontier model is getting wider, not narrower, because frontier labs are scaling compute, data, and alignment research simultaneously.

Part 5: The Models

Tier	Model	Role	Cost (in/out)
Fable	Fable 5.1	The ceiling	$10/$50
Opus	Opus 5	The heavy lifter	$5/$25
Sonnet	Sonnet 5	The workhorse	$2–3/$10–15
Haiku	Haiku 4.5	The sprinter	$0.80/$4

Fable 5.1

Most capable. Multi-hour coding, scientific research, long-form writing, creative fiction, financial analysis, defensive cybersecurity. 1M context, 128K output. Your escalation model.

Opus 5

Recommended for hard work. Half Fable's price. Production code, olympiad math, novel reasoning (4× any competitor), computer use, deep document analysis. Where 80% of hard work happens.

Sonnet 5

Everyday default. Cheapest capable model. Coding, chat, analysis, content, automation. Beats Opus on some benchmarks. Start here.

Haiku 4.5

Fastest, cheapest. 200K context. Classification, tagging, extraction, sub-tasks. Volume work.

Legacy

Fable 5, Opus 4.8/4.7/4.6, Opus 3, Sonnet 4.6 — still available, all superseded.

Side-by-Side

	Fable 5.1	Opus 5	Sonnet 5	Haiku 4.5
Context	1M	1M	1M	200K
Output	128K	128K/300K	128K	Standard
Thinking	Always on	Always on	Yes	No
Coding	★★★★★	★★★★★	★★★★	★★
Research	★★★★★	★★★★	★★★	★
Creative	★★★★★	★★★★	★★★	★★
Speed	★★	★★★	★★★★	★★★★★
Cost	★★	★★★	★★★★	★★★★★

Who Should Use What

New to AI

Sonnet 5 or Opus 5. Ask anything.

Business owner

Sonnet 5 daily. Opus 5 for deep work. Fable 5.1 for hardest tasks.

Engineer

Opus 5 main. Fable 5.1 for massive codebase analysis.

Writer

Fable 5.1 for best creative work. Sonnet 5 for everyday writing.

Researcher

Opus 5 mostly. Fable 5.1 for deep multi-step research.

Building a product

Mix models. Haiku for volume. Sonnet for the middle. Opus for judgment.

The Bottom Line

For the non-believers: this is not a chatbot and it is not hype. Under the hood it is hundreds of billions of learned parameters in a Transformer that computes attention across a million tokens. Trained on more text than any human will read. Fine-tuned with a published constitution. Built by a team that left the top AI lab in the world because they believed safety mattered more than speed.

It solves olympiad math. Writes production software. Explains financial crashes that stumped human teams for years. Operates a computer. It is not perfect — it makes mistakes, it has no consciousness, verify it on high-stakes decisions. But it is the most capable general-purpose tool that has ever existed, and it is getting measurably better every few months.

Sonnet 5 → Opus 5 → Fable 5.1. Step up only when the lower tier still falls short.