Build Your First LLM

A Hands-On Guide to Language Models

Hasan Degismez

2025-12-26

Build Your First LLM

A Hands-On Guide to Language Models

Hasan Degismez

Table of contents

Preface

> "The impediment to action advances action. What stands in the way becomes the way."
> — **Marcus Aurelius**, *Meditations*

Every concept that seems foreign today will become familiar through patient effort. What feels difficult now will feel natural by the end. You don't need to be extraordinary. You need to begin.

This book will teach you to build a Large Language Model from scratch. Not just use AI, but understand it, construct it, make it work.

If you're curious about how AI works, this book is for you. We assume no prior programming experience and no advanced math. We start from first principles and build up, piece by piece, explaining everything along the way.

All you need is curiosity, and the willingness to work through each chapter, one step at a time.

What You'll Build

By the end of this book, you'll have constructed a working language model. You'll understand how it processes text, learns patterns, and generates responses. More importantly, you'll understand *why* it works, not just that it does.

How to Use This Book

Every chapter includes a companion notebook that runs in Google Colab. No installation, no setup on your computer. Just click and start coding.

Each chapter ends with review questions to test your understanding. You'll find all the answers in Appendix D, so you can check your work and fill any gaps before moving on.

How This Book Is Organized

Part I: Foundations covers what AI and LLMs actually are, how machines process language, and the innovations that made this possible.

Part II: Python Essentials teaches the programming tools you'll need, nothing more.

Part III: Build Your First LLM is the core of the book. We construct a language model from scratch, step by step.

Part IV: Make It Useful covers fine-tuning, prompting, and putting your model to work.

Part V: Share It With the World shows you how to deploy your creation so others can use it.

Our Approach

We follow a few key principles:

- **Concepts before code**: Understand *why* before *how*
- **Analogies for everything**: Complex ideas explained through familiar comparisons
- **No skipped steps**: Every line of code explained
- **One continuous project**: Build one thing, from start to finish

Understanding comes through doing. The concepts will make sense because you'll build them yourself.

Copyright

First Edition: 2025

Visit the companion website: llm.degismez.com

Part I

Part I: Foundations

Chapter 1

What is AI, Really?

> "A computer would deserve to be called intelligent if it could deceive a human into believing that it was human." — **Alan Turing**, Computer Scientist

---

What You'll Learn - Why movie AI has nothing to do with real AI - What AI actually is (hint: pattern recognition at scale) - How LLMs fit into the broader AI landscape - What LLMs can and cannot do (and why they make mistakes) - Why building your own LLM matters

Key Terms

- *Artificial Intelligence (AI)*: Computer systems that perform tasks typically requiring human intelligence See glossary
- *Machine Learning (ML)*: AI systems that learn from data rather than explicit programming See glossary
- *Deep Learning (DL)*: ML using neural networks with many layers See glossary
- *Large Language Model (LLM)*: Deep learning systems trained on vast text to generate and understand language See glossary
- *Parameters/Weights*: The numbers in an AI model that get adjusted during training See glossary
- *Training*: The process of adjusting model parameters using data to improve performance See glossary
- *Emergence*: Complex capabilities arising from simple training objectives at sufficient scale See glossary
- *Hallucination*: When an LLM generates false information confidently See glossary

Checkpoint

By the end of this chapter, you'll understand:

1. The difference between movie AI and real AI (pattern recognition vs. consciousness)
2. How AI learns by adjusting numbers based on feedback
3. Where LLMs fit in the AI hierarchy (AI → ML → DL → LLM)

4. What LLMs can do (write, code, reason) and what they cannot (access real-time info, calculate reliably)
5. Why hallucination happens (lossy compression of training data)
6. Why building your own LLM deepens your understanding and career prospects

---

You've probably heard a lot about AI lately. It's in the news, it's on social media, your coworker won't stop talking about ChatGPT. Maybe you've used it yourself and thought: *How does this thing actually work?*

That's what this book will teach you. Not just how to use AI, but how to build it. By the end, you'll have created your own language model from scratch, the same fundamental technology behind ChatGPT, Claude, and every other AI assistant making headlines.

But before we write a single line of code, we need to understand what we're actually building. And that starts with clearing up some misconceptions.

1.1 Forget the Movies

Let's get something out of the way: almost everything Hollywood has taught you about AI is wrong.

In the movies, AI looks like this: - A robot with glowing red eyes plotting humanity's destruction (Terminator) - A calm voice slowly going insane and killing astronauts (HAL 9000) - A charming humanoid who falls in love and questions the nature of consciousness (Ex Machina)

These make for great stories. They also have almost nothing to do with real AI.

Here's what AI is *not*: - **Conscious.** Current AI shows no evidence of feelings, desires, or experiences. It doesn't "want" anything. - **Sentient.** There's no evidence of inner life. No thoughts when it's turned off. No dreams. - **Plotting against us.** Current AI systems don't have the kind of conscious scheming portrayed in films (though AI alignment, making sure AI does what we intend, remains an active research area). - **Magic.** Everything AI does follows from math, data, and computation. No mysteries.

The gap between movie AI and real AI is roughly the gap between a dragon and a lizard. Yes, they're both reptiles. But one breathes fire and hoards gold, while the other eats crickets and sits on a rock.

Movie AI vs. Real AI

Movie AI	Real AI
Has consciousness and feelings	Shows no evidence of inner experience
Wants to take over the world	Wants nothing; it's a tool
Thinks like a human, but faster	Processes patterns in data
Can do anything once "activated"	Can only do what it was trained for
Runs on mysterious future technology	Runs on GPUs doing matrix multiplication
Is either humanity's savior or destroyer	Is a very sophisticated autocomplete

That last one might seem like an insult to AI, but it's not. "Sophisticated autocomplete" is genuinely impressive. The fact that predicting the next word can produce coherent essays, working code, and creative stories is one of the most surprising discoveries in computer science.

But it's still pattern matching. Very good pattern matching.

> "The question of whether a computer can think is no more interesting than the question of whether a submarine can swim." — **Edsger W. Dijkstra**, Computer Scientist

Dijkstra was skeptical of AI, but his analogy is instructive: submarines don't swim like fish, yet they navigate water effectively. Similarly, AI doesn't think like humans, but it processes information in useful ways. Arguing about definitions misses the point.

1.2 AI is Pattern Recognition

So if AI isn't a thinking machine, what is it?

At its core, AI is **pattern recognition at scale**. It finds patterns in data that are too complex or too numerous for humans to find manually.

Let's make this concrete with something you use every day: a spam filter.

1.2.1 The Spam Filter Example

Every time you check your email, AI is working for you. Your spam filter doesn't follow a list of rules someone wrote by hand ("if email contains 'Nigerian prince', mark as spam"). Instead, it learned from millions of examples.

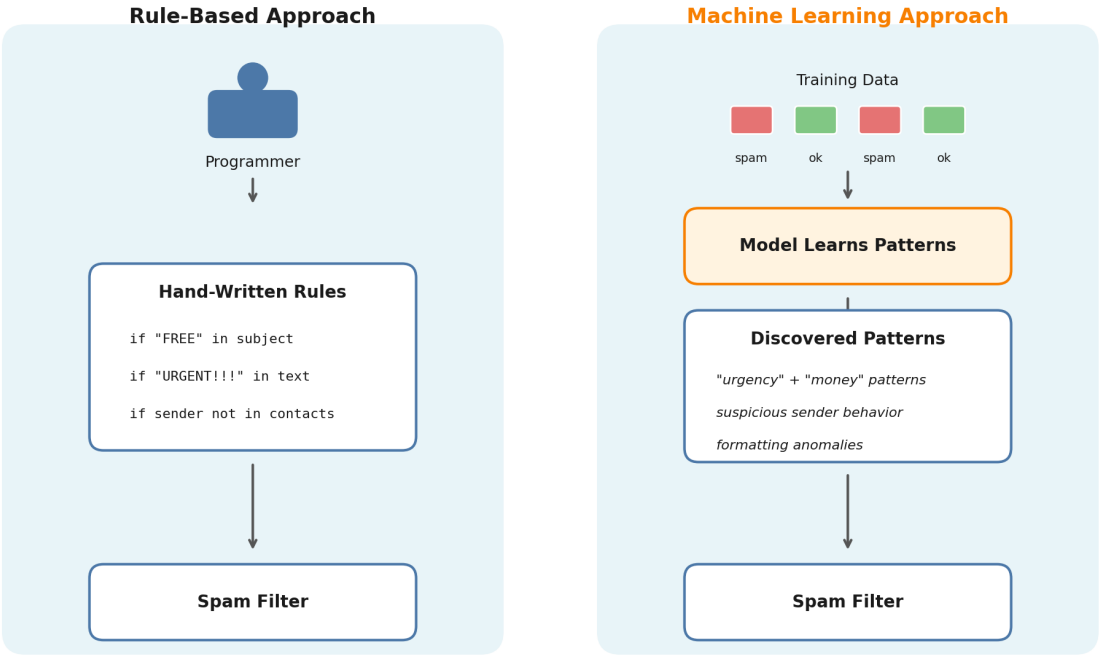

Figure 1.1: Rule-based vs machine learning: Instead of hand-written rules, the ML approach learns patterns from labeled examples.

Here's how:

1. **Training data**: The filter was shown millions of emails, each labeled "spam" or "not spam" by humans.
2. **Finding patterns**: The system discovered patterns on its own:
   - Spam emails often have ALL CAPS WORDS
   - Spam often mentions money, prizes, or urgency
   - Spam often comes from certain types of addresses
   - Spam often has specific formatting patterns
3. **Making predictions**: When a new email arrives, the filter checks it against all these learned patterns and predicts: spam or not?

The key insight: **nobody programmed these rules**. The system discovered them by looking at examples. That's what makes it "learning."

Think of it like training a new employee to sort mail. You don't give them a 500-page rule book. You sit with them, go through a stack of mail, and say "this is junk, this is real, this is junk…" Eventually, they get it. They've learned the patterns.

AI works the same way, except it can learn from millions of examples instead of hundreds, and it never gets tired or distracted.

1.2.2 The Formula for AI

Here's the secret formula behind most modern AI:

AI = Statistics + Lots of Data + Computing Power

That's it. No magic, no consciousness, no secret sauce.

- **Statistics**: Mathematical techniques for finding patterns and making predictions
- **Lots of Data**: Millions or billions of examples to learn from
- **Computing Power**: Fast processors that can crunch through all that data

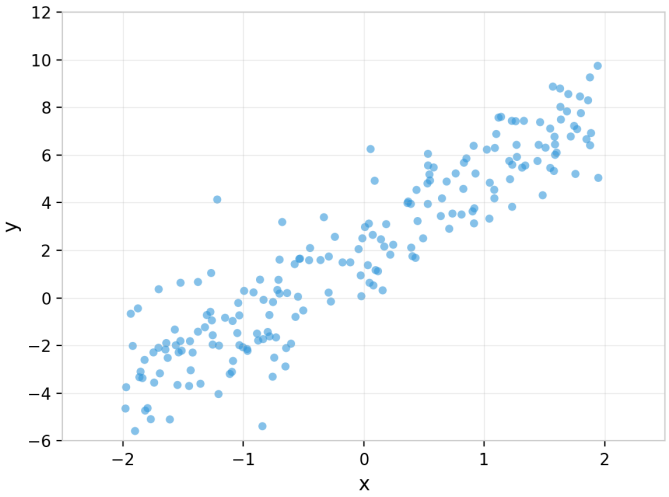

Figure 1.2: Visualizing data: points with a clear pattern (a line) but some noise.

Twenty years ago, we had the statistics. We had some data. But we didn't have enough computing power to make it work at scale. Now we do. That's why AI suddenly seems to be everywhere.

1.2.3 What "Learning" Actually Means

When we say an AI system "learns," we mean something specific: it adjusts numbers based on feedback.

Every AI model is, at its heart, a big collection of numbers (called "parameters" or "weights"). These numbers start random.

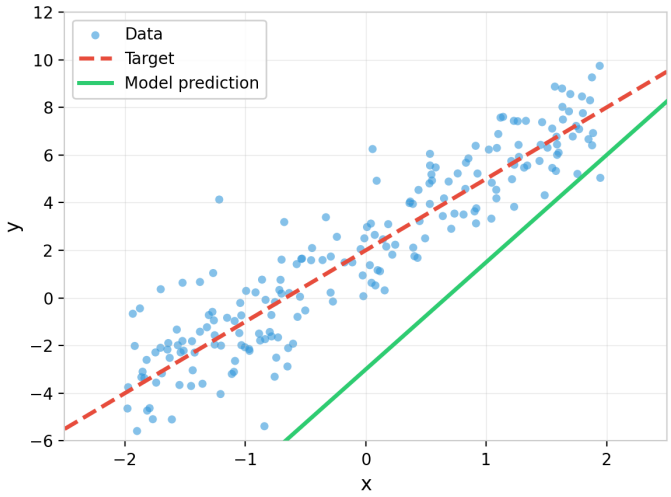

Figure 1.3: Before training: The model (green line) guesses randomly and misses the data (blue dots).

During training, the system:

1. Makes a prediction
2. Checks how wrong it was
3. Adjusts its numbers slightly to be less wrong next time
4. Repeats billions of times

In our line-fitting example, there are just two numbers to adjust: the **weight** (how steep the line is) and the **bias** (where the line crosses the vertical axis). The chart below shows these two numbers changing over 10 training steps:

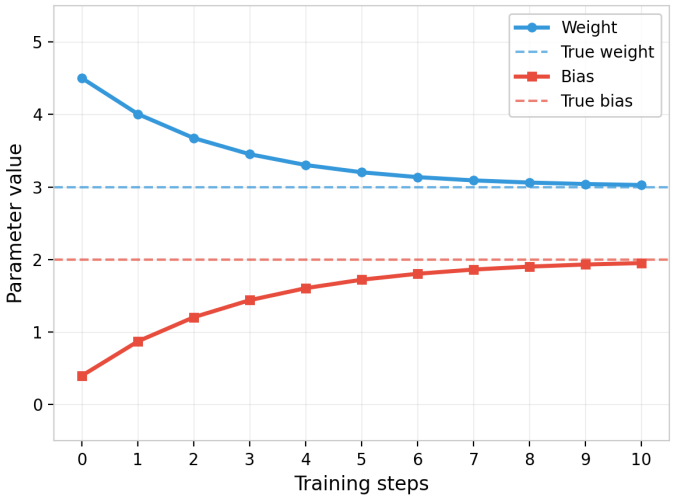

Figure 1.4: Learning in action: The model adjusts its weight and bias over training steps to find the best fit.

As the weight drops from 4.5 toward 3 and the bias rises from 0.4 toward 2, the green prediction line rotates and shifts until it matches the target:

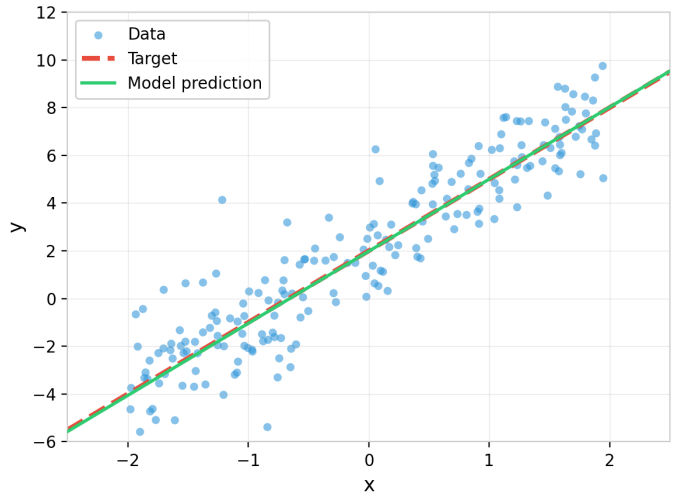

Figure 1.5: After training: The model has found the pattern and fits the data.

That's learning. It's not philosophical. It's not mysterious. It's just adjusting numbers based on errors until the numbers produce good outputs.

Modern LLMs have hundreds of billions (sometimes over a trillion) of these numbers. They were

adjusted through trillions of predictions on massive amounts of text. The result is a system that can predict the next word in almost any context with remarkable accuracy.

1.3 The AI Family Tree

"AI" is a broad term that covers many different technologies. Let's sort them out.

1.3.1 The Hierarchy

Think of it as nested circles, each one inside the next:

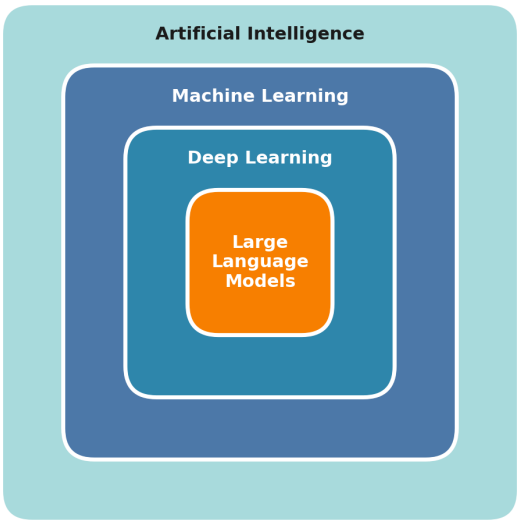

Figure 1.6: The nested hierarchy of AI, Machine Learning, Deep Learning, and Large Language Models.

Let's define each layer:

Artificial Intelligence (AI): The broadest category. Any computer system that performs tasks we'd consider "intelligent" if a human did them. This includes your spam filter, chess programs, recommendation systems, and yes, ChatGPT.

Machine Learning (ML): A subset of AI where systems learn from data rather than following hand-coded rules. Instead of a programmer writing "if X then Y," the system discovers its own

patterns from examples.

Deep Learning (DL): A subset of ML using "neural networks," systems loosely inspired by how neurons in the brain connect. "Deep" means these networks have many layers, allowing them to learn increasingly abstract patterns. For example, in image recognition, early layers detect lines and curves, middle layers recognize shapes like eyes, and deep layers conclude "this is a face." Deep learning powers image recognition, speech processing, and many other applications. LLMs are the branch focused on language.

Large Language Models (LLMs): A subset of deep learning specifically designed for language. These models are trained to predict the next word in a sequence, and they're "large" because they have billions of parameters and are trained on vast amounts of text.

1.3.2 A Brief Timeline

AI has a long history with dramatic ups and downs:

Year	Event
1950	Alan Turing proposes the "Turing Test" for machine intelligence
1956	"Artificial Intelligence" term coined at Dartmouth Conference
1960s-70s	Early optimism, then first "AI Winter" when progress stalled
1997	IBM's Deep Blue defeats world chess champion Garry Kasparov
2012	AlexNet wins ImageNet competition, sparking deep learning revolution
2017	Google publishes "Attention Is All You Need," the Transformer architecture
2018	GPT-1 released (117 million parameters)
2020	GPT-3 released (175 billion parameters), capabilities surprise researchers
2022	ChatGPT launches, AI enters mainstream consciousness
2023+	Rapid advancement continues, with new models releasing every few months

The 2017 Transformer paper is particularly important for this book. That architecture is what you'll learn to build. It's the foundation of every modern LLM.

1.4 What Makes LLMs Special?

Language has always been hard for computers. Why? Because language is ambiguous, contextual, and constantly evolving.

Consider the sentence "I saw her duck": - Did you see a bird she owns? - Did you watch her lower her head suddenly?

Humans resolve this ambiguity instantly using context. Computers traditionally struggled because they had no way to represent context.

LLMs solved this problem. Using the Transformer architecture (which you'll learn in Chapters 3 and 4), they can consider entire paragraphs of context when processing each word. This lets them handle ambiguity, follow references, and maintain coherence across long passages.

1.4.1 What LLMs Can Do

Modern LLMs are remarkably versatile:

- **Write and edit text**: Essays, emails, reports, creative fiction
- **Generate code**: Working programs in dozens of languages
- **Translate languages**: Handling many language pairs well, though quality varies
- **Summarize documents**: Condensing long texts into key points
- **Answer questions**: Drawing on vast training knowledge
- **Reason through problems**: Multi-step logic (though not always reliably)
- **Adopt personas**: Following instructions to behave differently
- **Analyze and explain**: Breaking down complex topics

The remarkable thing is that nobody explicitly programmed any of these capabilities. They emerged from one simple training objective: predict the next word.

Researchers call this "emergence": when simple rules produce complex behavior. Nobody fully understands why predicting the next word leads to what looks like reasoning, and there's active debate about whether it's genuine reasoning or very sophisticated pattern matching. It's one of the most fascinating open questions in the field.

1.4.2 What LLMs Cannot Do (On Their Own)

Here's something important to understand: the LLM itself is just one component. When you use ChatGPT or Claude, you're using a *product* built around an LLM, with additional tools layered on top.

The base LLM cannot: - **Access real-time information**: It only knows what was in its training data - **Remember previous conversations**: Each session starts fresh - **Perform actions in the world**: It only generates text - **Do reliable math**: It pattern-matches rather than calculates - **Know when it's wrong**: It has no self-awareness

But modern AI products add tools to overcome these limitations: - **Web search**: ChatGPT can search the internet for current information - **Code execution**: It can run Python to do actual calculations - **Memory systems**: Some products remember details across conversations - **Document retrieval**: It can search through uploaded files or databases

This distinction matters. When ChatGPT gives you today's weather, the LLM isn't "knowing" the weather. It's using a tool to look it up, then reporting what it found. The LLM is the brain; the tools are like giving it hands, eyes, and a phone.

In Part 4 of this book, you'll learn to add these capabilities yourself using techniques like RAG (Retrieval-Augmented Generation) and tool integration.

The Core LLM vs. The Product

The LLM Itself	Tools Added By Products
Predicts text based on training	Web search for current info
No memory between sessions	Memory systems for continuity
Cannot calculate reliably	Code interpreter for math

The LLM Itself	Tools Added By Products
Cannot verify facts	Document retrieval for accuracy
Cannot take actions	API integrations for actions

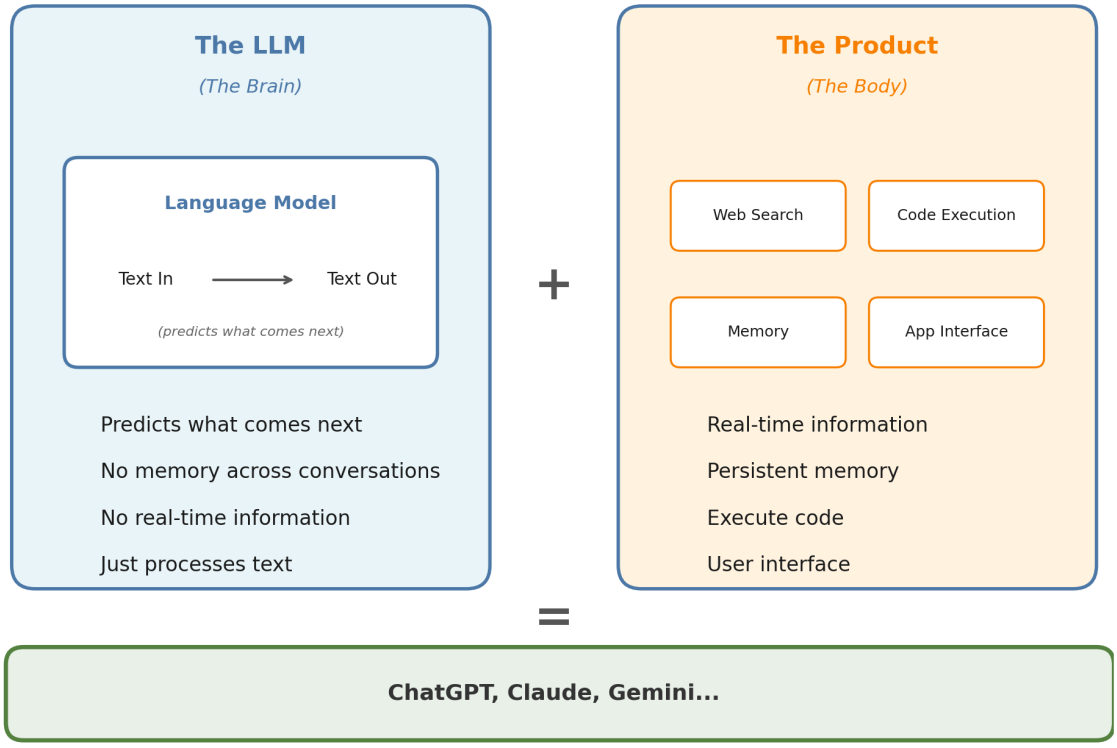

Figure 1.7: The LLM predicts text; the product adds tools, memory, and interface. Together they become ChatGPT, Claude, or Gemini.

1.4.3 Why Do LLMs "Hallucinate"?

You've probably heard that LLMs sometimes make things up, stating false information with complete confidence. This is called "hallucination."

Why does it happen? Because LLMs don't have a database of facts. They have patterns in text.

The "Lossy Compression" Analogy

Andrej Karpathy, a founding member of OpenAI and former Director of AI at Tesla, describes an LLM as a "lossy compression" of the internet.

Imagine taking the entire internet and trying to zip it into a tiny file. You can't keep every word exactly as it was; there's not enough space. Instead, you keep the *gist*, the patterns, the general ideas. - **Compression**: The model learns the general rules (grammar, reasoning, facts about the world). - **Lossy**: It loses the exact details.

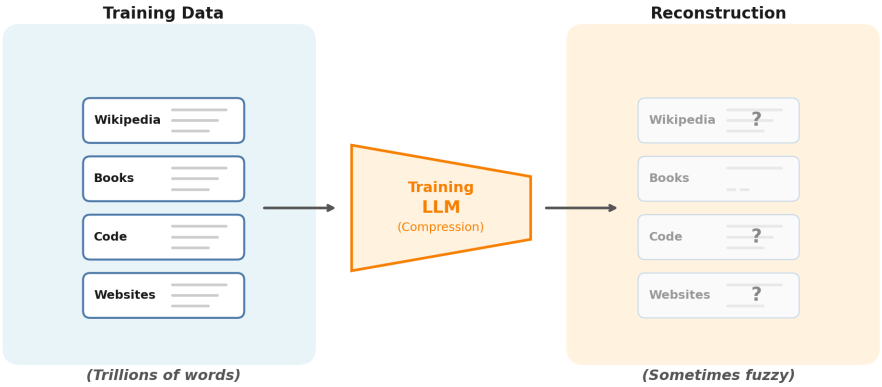

Figure 1.8: Lossy Compression: Squeezing the internet into a model results in a "fuzzy" reconstruction.

When you ask it a question, it's "unzipping" that compressed knowledge on the fly. Usually, it reconstructs the information correctly. But sometimes, because the compression was "lossy," the reconstruction is slightly off. It fills in the gaps with what *looks* plausible based on the patterns it saved. That's a hallucination: a reconstruction error from a fuzzy memory.

> "All models are wrong, but some are useful." — **George Box**, Statistician

Box was talking about statistical models, but it applies perfectly to LLMs. They're models of language, sophisticated ones, but models nonetheless. They don't capture truth; they capture patterns that often correlate with truth. That's why they're useful. It's also why you should verify anything important.

1.5 Hands-On Exercises

Reading about AI only takes you so far. Let's get hands-on.

If you have access to ChatGPT (or Claude, or another LLM), try these prompts and observe what happens. There are no wrong answers; you're exploring.

1.5.1 Prompts to Try

1. Simplification

```
Explain photosynthesis like I'm 5 years old.
```

Watch how it adapts vocabulary and uses analogies.

2. Creative writing

```
Write a haiku about debugging code.
```

Notice it follows the 5-7-5 syllable structure.

3. Role-playing

```
You are a pirate captain. Explain how compound interest works.
```

See how it maintains the persona while explaining a complex topic.

4. Reasoning

```
If it takes 5 machines 5 minutes to make 5 widgets,
how long would it take 100 machines to make 100 widgets?
```

This is a classic trick question. Does the LLM get it right?

5. Tool use test

```
What major news events happened yesterday?
```

If the LLM has web search, watch it look up current information. If not, it should refuse or admit it doesn't know. This shows the difference between the base model and added tools.

6. Self-awareness

```
What are you unable to do? Be honest about your limitations.
```

See how it describes its own constraints.

1.5.2 Exercises

Exercise 1: Find a Mistake

Ask the LLM about a topic you know well. Can you find something it gets wrong? It might be a subtle error or a complete fabrication. This isn't to criticize the technology. It's to build your intuition about when to trust AI and when to verify.

Exercise 2: Variation

Ask the same question three times in three separate conversations. How different are the responses? This helps you understand that LLMs are probabilistic: they don't give fixed answers.

Exercise 3: Push the Limits

Try to find tasks the LLM struggles with: - Complex multi-step math (try without code execution if possible) - Very obscure or specialized topics - Tasks requiring precise factual accuracy - Logical puzzles designed to trip up pattern-matching

Document what you find. Where does it excel? Where does it fail? If the product has tools like web search or code execution, try the same tasks with and without those tools to see the difference.

> "Tell me and I forget. Teach me and I remember. Involve me and I learn." — **Benjamin Franklin**

1.6 Why Learn to Build One?

You might be wondering: if ChatGPT already exists, why learn to build an LLM?

It's a fair question. But consider this: mechanics understand cars better than drivers. Pilots who understand aerodynamics make better decisions in emergencies. Chefs who understand food chemistry create better dishes than those who just follow recipes.

Understanding the internals changes how you interact with a technology. Here's why it matters for AI:

1.6.1 Demystification

Right now, AI probably feels like magic to you. By the end of this book, it won't. You'll understand every component, every mathematical operation, every training step.

This matters because magic invites fear and hype. Understanding invites good judgment.

Imagine you're in a meeting, and someone proposes using AI for a project. Everyone else nods along, unsure whether it's feasible or hype. But you know. You understand what LLMs actually can and can't do, where they excel and where they fail. You speak up with a realistic assessment based on how the technology actually works.

That's the difference between using AI and understanding it.

1.6.2 Career Advantage

The job market is changing fast. AI skills are in demand across industries: - Software engineers who understand ML systems - Product managers who can work with AI teams - Designers who understand AI capabilities and limitations - Researchers who can push the technology forward - Entrepreneurs building AI-powered products

You don't need to become a machine learning researcher to benefit. Understanding how LLMs work makes you more effective in almost any technical role.

But beyond job titles: the people who understand a transformative technology early gain an advantage that compounds over time. Those who understood the internet in 1995 built companies that reshaped industries. Those who understand AI now are in a similar position.

1.6.3 Customization

Off-the-shelf LLMs are general-purpose. But you might need something specific: - A model fine-tuned on your company's documents - A smaller model that runs on a phone - A specialized assistant for a particular domain - A system that works in a language or domain with limited training data

Understanding the fundamentals lets you customize, fine-tune, and adapt models to your needs rather than waiting for someone else to build what you need.

1.6.4 Understanding Through Building

There's something deeply satisfying about understanding how things work. When you ask ChatGPT a question and it responds intelligently, you'll know *exactly* what's happening under the hood. The mystery becomes mechanics.

That moment when you train your own model and watch it generate coherent text for the first time (text that came from mathematical patterns you helped create) is genuinely thrilling. It's the difference between watching a magic trick and knowing how it's done.

Richard Feynman, the legendary physicist, kept a note on his blackboard: "What I cannot create, I do not understand."

This philosophy is championed today by **Karpathy**, whose work has inspired thousands of engineers (including me, the author of this book). His "Zero to Hero" series is the gold standard for understanding neural networks from the ground up. By building an LLM yourself, you are following in this tradition: replacing mystery with mastery.

> **Go Deeper**: If you want to see the raw code implementation alongside this book, I highly recommend Andrej Karpathy's YouTube series "Neural Networks: Zero to Hero." This book is designed to be the perfect companion to those lectures, expanding on the concepts, adding more context, and guiding you through the practicalities of building a production-ready system.

1.7 Checkpoint Exercise

Time: 15-20 minutes

Instructions: Identify 5 AI systems you've interacted with in the past week. For each one:

1. Name the system/product
2. What does it do?
3. Where does it fit on the AI family tree (AI → ML → DL → LLM)?
4. What patterns is it recognizing?

Examples to get you started:

System	What it does	Category	Patterns recognized
Gmail spam filter	Sorts email	ML	Spam vs. legitimate email patterns
Netflix recommendations	Suggests shows	ML	Viewing preferences and similar users
Siri/Alexa	Voice assistant	DL (Classic) / LLM (Modern)	Speech patterns, intent from text

System	What it does	Category	Patterns recognized
Autocomplete on phone	Predicts next word	ML/DL	Word sequences in context

Now find 5 of your own!

1.8 Key Takeaways

What you learned:

1. AI is pattern recognition at scale: statistics + data + computing power
2. Machine learning systems learn by adjusting parameters based on feedback
3. LLMs fit into the hierarchy: AI → ML → Deep Learning → LLM
4. LLMs can write, code, translate, and reason, but cannot access real-time info or calculate reliably on their own
5. Hallucination happens because LLMs are "lossy compression" of training data
6. Modern AI products add tools (web search, code execution) around the base LLM

Key concepts:

- **Pattern recognition**: Finding patterns in data too complex for humans to manually code
- **Parameters/Weights**: The numbers adjusted during training (GPT-3 has 175 billion)
- **Emergence**: Complex capabilities arising from simple training objectives
- **Hallucination**: Reconstruction errors from compressing vast knowledge into patterns
- **Base LLM vs Product**: The model predicts text; tools give it real-world capabilities

The AI hierarchy:

```
Artificial Intelligence (broadest)
       Machine Learning (learns from data)
           Deep Learning (neural networks with many layers)
               Large Language Models (predicts next word)
```

> **Review Question Answers**
>
> All answers to these review questions are available in Appendix D.

1.9 Review Questions

1. What is the simplest definition of AI?
2. How does Machine Learning differ from traditional programming?

3. Where do LLMs fit in the AI family tree?
4. Name three things LLMs are good at and three things they cannot do well.
5. Why does hallucination happen in LLMs?
6. Why might understanding how LLMs work be valuable, even if you don't plan to become an AI researcher?

1.10 What's Next

Now you understand what AI is and isn't. You know where LLMs fit in the landscape and what makes them special.

But we've been talking at a high level. How does a computer actually work with text? It only understands numbers. How do we get from words to numbers in a way that captures meaning?

That's Chapter 2: How Computers "Understand" Words. We'll explore how text becomes math, the foundation everything else builds on.

Chapter 2

How Computers "Understand" Words

> "The limits of my language mean the limits of my world." — **Ludwig Wittgenstein**, Philosopher

---

What You'll Learn - Why simple approaches to representing words (like ASCII) fail - How words can be represented as positions in a mathematical space - What embeddings are and why they're powerful - Why "king - man + woman = queen" actually works - How machines learn word meanings from context

Key Terms

- *Embedding*: A representation of a word as a vector of numbers that captures its meaning See glossary
- *Vector*: A list of numbers representing a position in space
- *Dimension*: One axis in the embedding space; more dimensions allow more nuanced representations See glossary
- *Word Space*: The mathematical space where word embeddings live; distances correspond to semantic differences
- *Distributional Hypothesis*: The idea that words appearing in similar contexts have similar meanings
- *Semantic Similarity*: How related two words are in meaning; measurable as distance in embedding space
- *Cosine Similarity*: The standard method for measuring how similar two vectors are; ranges from -1 to 1 See glossary
- *Word2Vec*: A foundational 2013 system for learning word embeddings from text
- *Self-Supervised Learning*: Training where the learning signal comes from the data itself, not human labels

Checkpoint

By the end of this chapter, you'll understand:

1. How words become numbers that capture meaning
2. Why embeddings use hundreds of dimensions
3. How context teaches machines about semantic relationships
4. How to see embeddings in action
5. Why attention is needed to combine embeddings into contextual understanding

---

In Chapter 1, we established that LLMs predict the next word. But that raises an immediate question: computers only understand numbers. They can't read "cat" or "democracy" or "beautiful," they can only process 0s and 1s.

So how do we get from words to numbers in a way that captures *meaning*?

This chapter answers that question. The solution is one of the most elegant ideas in all of AI, and understanding it is essential for everything that follows.

2.1 The Computer's Dilemma

Imagine you're trying to teach a friend who only speaks numbers. You want to explain the concept of a "cat." How do you do it?

Your first instinct might be to convert letters to numbers. After all, that's how computers store text.

2.1.1 The ASCII Approach (And Why It Fails)

Every character on your keyboard has a number. The letter 'c' is 99, 'a' is 97, 't' is 116. So "cat" becomes [99, 97, 116].

Problem solved? Not even close.

Consider: - "cat" = [99, 97, 116] - "car" = [99, 97, 114]

These numbers are almost identical (they differ by just 2 in the last position). But a cat and a car have nothing in common. One is a living animal; the other is a machine with wheels.

Meanwhile: - "cat" = [99, 97, 116] - "kitten" = [107, 105, 116, 116, 101, 110] - "feline" = [102, 101, 108, 105, 110, 101]

These numbers look completely different, even though all three words refer to essentially the same creature.

ASCII codes capture *spelling*, not *meaning*. And spelling is arbitrary: there's no reason "cat" is spelled c-a-t rather than something else.

2.1.2 The Word ID Approach (And Why It Also Fails)

Okay, forget letters. What if we just assign each word a unique ID?

- cat = 1
- dog = 2
- fish = 3
- feline = 4
- kitten = 5
- automobile = 6

Now "cat" and "car" have different numbers. But we've created a new problem: these numbers have no relationship to each other.

In this system, "cat" (1) and "kitten" (5) are just as unrelated as "cat" (1) and "automobile" (6). The numbers don't capture that cats and kittens are similar, while cats and automobiles aren't.

We could do math on these IDs: cat + dog = 3. But that equals fish, mathematical nonsense that reveals how poorly these IDs capture relationships.

2.1.3 What We Actually Need

Here's what we need: a way to convert words to numbers where:

1. **Similar words get similar numbers**
2. **Unrelated words get different numbers**
3. **Relationships between words are preserved**

This turns out to be possible. But it requires thinking about numbers differently.

2.2 Words as Locations

Here's the key insight: instead of giving each word a single number, give each word a *position*.

Think of a city map. On a map, every location has coordinates: an X position and a Y position. Two restaurants near each other have similar coordinates. A restaurant and a park on opposite sides of town have very different coordinates.

What if we did the same thing for words?

2.2.1 Word Space

Imagine a two-dimensional space, like a piece of graph paper. Now imagine placing words on it based on their meaning:

In this visualization:

- **Gender Axis (X-axis)**: Words shift from masculine (left) to feminine (right). Notice how "man" and "woman" are far apart on this axis, but at similar heights.
- **Age Axis (Y-axis)**: Words shift from young (bottom) to old (top). "Boy" is at the bottom, "man" is in the middle, and "grandfather" is at the top.

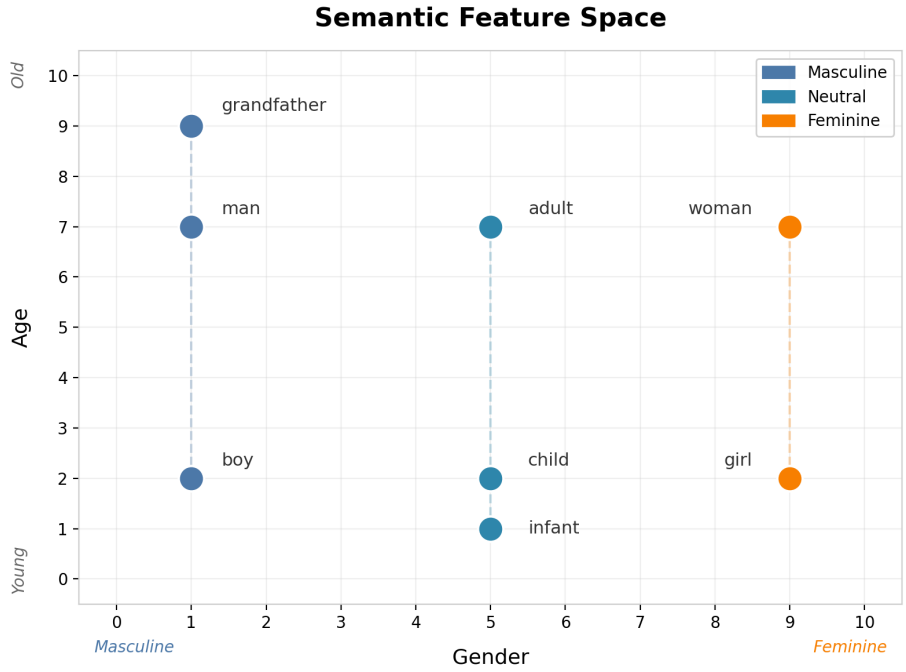

Figure 2.1: Words plotted in a 2D semantic space, where position reflects meaning. Similar words cluster together.

- **Clustering**: Words with similar semantic features end up in similar positions. "Boy" and "girl" sit at the same height (same age) but on opposite sides of the gender axis. "Man" and "woman" share the same pattern: same age, different gender.

(Note: In this simplified visualization, we've mapped words onto interpretable dimensions like Gender and Age. Real embeddings use hundreds of dimensions that emerge automatically from training data. These dimensions rarely correspond to neat human concepts; they capture statistical patterns that are mathematically useful but often impossible to name.)

The *position* of a word captures something about its meaning. Similar meanings lead to similar positions.

2.2.2 Coordinates Capture Meaning

In a 2D space, each word has two numbers (its X and Y coordinates):

- "boy" might be at position [1.0, 2.0] (Gender=1, Age=2)
- "man" might be at position [1.0, 7.0] (Gender=1, Age=7)
- "woman" might be at position [9.0, 7.0] (Gender=9, Age=7)

> **i** Math Note: What Is a Vector?
>
> A **vector** is just a list of numbers representing a position in space. The list `[1.0, 2.0]` is a 2-dimensional vector. Don't worry about the math details; if you want to understand vectors more deeply, see Appendix C.1 (Math Refresher: Scalars, Vectors, and Matrices).

The key insight: **the closer two words are in this space, the more similar their meanings**.

We can measure this mathematically. In this 2D grid, "boy" is closer to "man" (distance 5) than to "woman" (distance approx 9.4).

In real high-dimensional embeddings, the standard approach uses **cosine similarity**, which measures the angle between vectors. Two words pointing in similar directions are similar.

> **i** Math Note: How Does Cosine Similarity Work?
>
> **Cosine similarity** measures how similar two vectors are by looking at the angle between them, not their absolute distance. This makes it perfect for comparing word meanings. If you want to see the actual calculation, check Appendix C.4 (Math Refresher: Dot Products and Similarity).

Now we have numbers that capture meaning relationships.

2.2.3 But Two Dimensions Aren't Enough

Of course, two dimensions can't capture all the nuances of language. Consider these words:

- "king" and "queen" (both are royalty)
- "king" and "man" (both are male)

- "king" and "president" (both are leaders)

A king is similar to a queen in one way (royalty), similar to a man in another way (gender), and similar to a president in yet another way (leadership). To capture all these different kinds of similarity, we need more dimensions.

What if we had hundreds of dimensions? Or thousands?

2.2.4 Visualizing Higher Dimensions

It's hard to imagine 768 dimensions, but we can try to visualize 3 dimensions. By adding a third axis (Royalty), we can capture another type of relationship between words.

(Note: In this 3D diagram, we use color differently than in the 2D diagram above. Here, color encodes royalty level, not gender.)

In this 3D visualization, notice how the color gradient encodes the third dimension: blue indicates low royalty (commoners like boy, girl, man, woman), while orange indicates high royalty (king, queen, prince, princess). The spatial clustering makes relationships visible: you can see that prince is closer to king than to boy, and princess is closer to queen than to girl. And this is with just three dimensions. You can't visualize 768 dimensions, but the math works the same way: words with similar meanings end up as neighbors in this high-dimensional space.

2.3 The Magic of Embeddings

This is exactly what modern AI does. Instead of 2 dimensions, words are represented in spaces with hundreds or thousands of dimensions.

This representation (a list of numbers that encodes a word's position in a high-dimensional space) is called an **embedding**.

2.3.1 What an Embedding Looks Like

Here's what a real embedding might look like (simplified):

```
"king"   = [0.82, 0.31, 0.91, -0.24, 0.55, 0.12, ...]  (768 numbers total)
"queen"  = [0.79, 0.33, 0.88, -0.19, 0.52, 0.15, ...]  (similar!)
"banana" = [-0.51, 0.89, 0.11, 0.63, -0.22, 0.77, ...] (very different!)
```

Notice that "king" and "queen" have similar numbers in each position, while "banana" has completely different numbers. That's because kings and queens are semantically related, while bananas aren't related to either.

Did You Know? Multilingual Magic

Multilingual LLMs discover something remarkable: words like "dog" (English) and "hund" (German) end up as neighbors in embedding space, without anyone telling the model they're related. How?

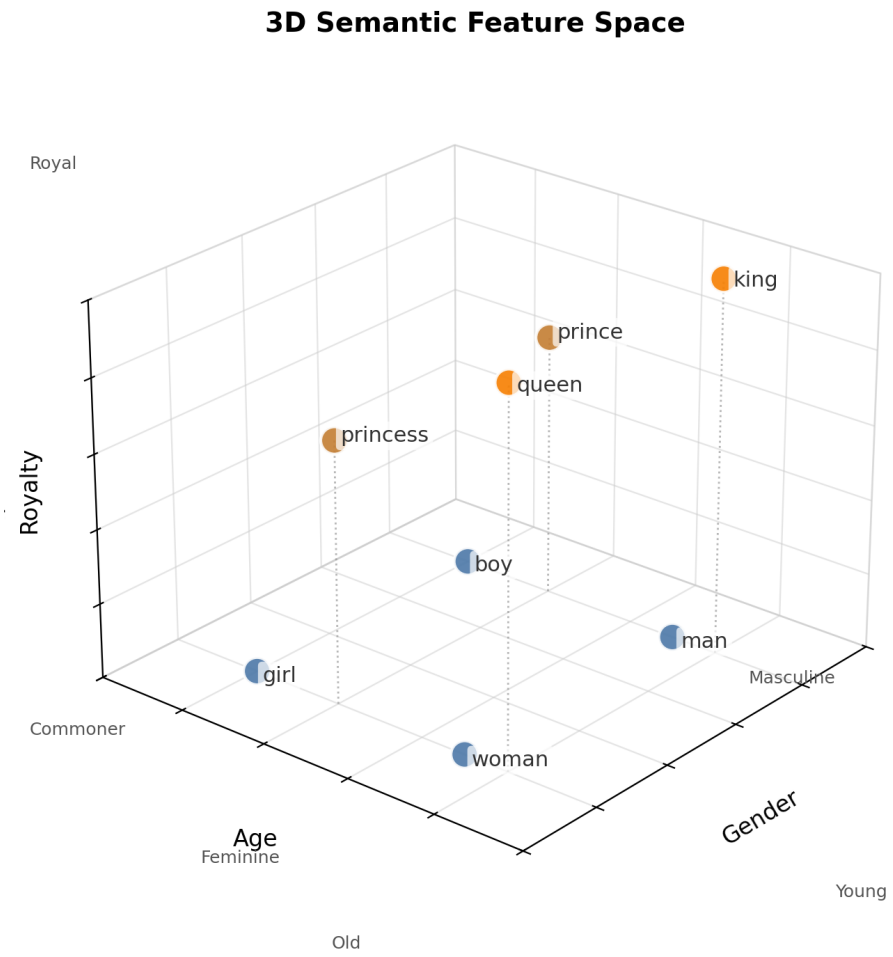

Figure 2.2: Adding a third dimension (Royalty) allows capturing more semantic relationships. King and queen share high royalty, while boy and girl remain commoners.

During training, these words appear in similar contexts (chasing cats, having four legs, barking), so the model learns they refer to the same concept. This means when an LLM learns a fact about dogs in English, it automatically "knows" it in German too. The embedding space organizes itself around meaning, not language.

2.3.2 Why More Dimensions Help

Think of it like describing a person:

- **1 dimension** (height): You can distinguish tall from short people.
- **2 dimensions** (height + weight): Now you can distinguish tall-thin from tall-heavy.
- **5 dimensions** (add age, hair color, eye color): Much more specific.
- **100 dimensions**: You can describe subtle differences in appearance.

Similarly:

- **2 dimensions** for words: Crude groupings (animals vs. objects)
- **100 dimensions**: Can distinguish mammals from reptiles, pets from wild animals
- **768 dimensions**: Can capture subtle nuances, like how "stroll" is a leisurely walk while "march" is a deliberate one

Modern LLMs use embedding dimensions ranging from 768 (smaller models) to 12,288 or more (large models). The number 768 is a common standard because it was the dimension used in the influential BERT-base and GPT-2 small models. Each dimension captures some aspect of meaning, though (importantly) no individual dimension has a clear human interpretation like "size" or "animacy." The dimensions emerge from training and encode meaning in distributed, complex ways.

2.3.3 Why "Embedding"?

The term comes from mathematics, where "embedding" means mapping objects from one space into another while preserving important structure. We're taking discrete symbols (words, where "cat" and "car" have no inherent relationship) and embedding them into continuous space (where their positions encode meaning).

Once words are embedded in this mathematical space, we can measure distances, find neighbors, and do arithmetic. This is what makes language processing mathematical, and therefore something computers can do.

2.4 The Famous Equation

In 2013, researchers at Google made an unexpected discovery. When they trained a system to create word embeddings, something remarkable emerged.

They found that you could do arithmetic with words.

2.4.1 King - Man + Woman = ?

Take the embedding for "king." Subtract the embedding for "man." Add the embedding for "woman." What do you get?

The result is a point in space, and the nearest vocabulary word to that point is… "queen."

```
nearest_word( embedding("king") - embedding("man") + embedding("woman") ) = "queen"
```

The computed vector isn't exactly equal to queen's embedding, but queen is the closest word in the vocabulary to that location.

Figure 2.3 shows this relationship geometrically. Notice how the purple arrows (the "gender direction" from man→woman and king→queen) are parallel. The same transformation that changes "man" to "woman" also changes "king" to "queen."

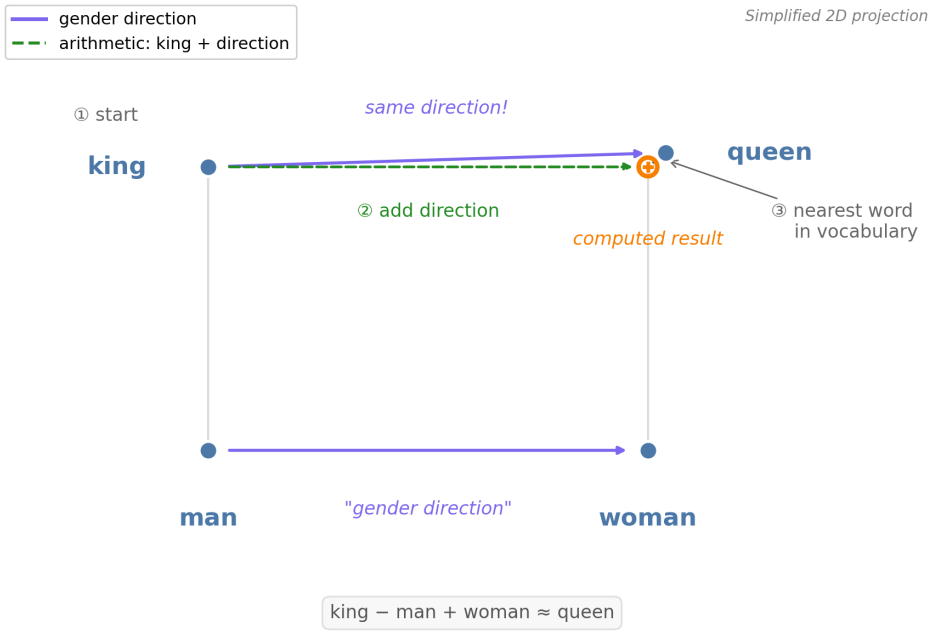

Figure 2.3: Vector arithmetic in embedding space: starting from "king" (blue dot), applying the same direction that transforms man→woman (purple arrow) produces a computed result (orange circle). The nearest vocabulary word to that point is "queen" (green dot).

Think about what this means. The system learned, without being told, that:

- The relationship between "king" and "man" (male royalty to male)
- Is the same as the relationship between "queen" and "woman" (female royalty to female)

Nobody programmed this. The embeddings *captured the underlying structure of language.*

2.4.2 More Examples

This works for many relationships (though not perfectly; accuracy on these "analogy tasks" is typically 40-70%, not 100%):

Capitals: - Paris - France + Italy ≈ Rome - Tokyo - Japan + Germany ≈ Berlin

Verb tenses: - walking - walk + swim ≈ swimming - ran - run + fly ≈ flew

Comparatives: - bigger - big + small ≈ smaller - fastest - fast + slow ≈ slowest

Relationships: - brother - man + woman ≈ sister - uncle - man + woman ≈ aunt

2.4.3 What This Reveals

This word arithmetic reveals something profound: embeddings don't just put similar words close together. They organize words so that *relationships* are consistent.

The "direction" from man to woman in embedding space is roughly the same as the direction from king to queen, from brother to sister, from uncle to aunt. The embeddings have learned an abstract concept of gender transformation.

Similarly, there's a direction for "capital of" that takes you from France to Paris, from Japan to Tokyo, from Germany to Berlin.

2.4.4 Important Caveats

This doesn't always work perfectly. Sometimes the nearest word isn't quite right. Sometimes biases in training data create problematic patterns. The arithmetic is approximate, not exact.

But the fact that it works at all (that you can do meaningful arithmetic with words) shows that embeddings capture something real about language structure. They're not just arbitrary numbers; they're a map of meaning.

> **i** Connection: Lossy Compression
>
> Remember the "lossy compression" analogy from Chapter 1? Embeddings are the *first* layer of that compression, taking the infinite complexity of human language and squeezing it into fixed-dimensional vectors. The entire model then continues this compression through attention and feedforward layers. Just as a compressed image loses some detail, the model captures patterns rather than exact facts. This is why LLMs can reason about concepts they've never seen exactly, and also why they sometimes "hallucinate" plausible-sounding but incorrect information.

2.5 How Are Embeddings Learned?

Where do these magical numbers come from? How does a system learn that "king" and "queen" should be near each other, while "king" and "banana" should be far apart?

The answer is beautifully simple: **context**.

2.5.1 You Shall Know a Word by the Company It Keeps

This phrase, from linguist J.R. Firth, captures a fundamental insight: words that appear in similar contexts tend to have similar meanings.

Consider these sentences: - "The _____ sat on the mat." - "The _____ chased the mouse." - "She stroked the _____'s fur."

What words fit in the blanks? Cat, dog, kitten, puppy. These words appear in similar contexts, so they must have related meanings.

Now consider:

- "I deposited money at the _____."
- "The _____ approved my loan."
- "She works as a teller at the _____."

The answer is "bank" (the financial institution). Different contexts than the cat sentences, different meaning cluster.

2.5.2 Learning from Prediction

Here's how embedding systems actually learn. The basic idea (pioneered by Word2Vec in 2013):

1. **Take a massive amount of text** (billions of words from the internet, books, etc.)
2. **Create a prediction task**: Given the words around a blank, predict the missing word
3. **Adjust the embeddings** so that words in similar contexts get similar positions

For example, given "The cat sat on the ***", the system should predict "mat," "floor," "couch," etc. are likely. Given "The bank approved the*** ", it should predict "loan," "mortgage," "application."

Through billions of these predictions, the system learns:

- "Cat" and "dog" appear in similar contexts → they get similar embeddings
- Words appearing in diverse contexts (like "bank" near both "river" and "money") end up with compromise embeddings, a limitation we'll address shortly
- "King" appears in contexts about royalty, leadership, male rulers → its embedding encodes all of this

2.5.3 The Beauty of Self-Supervision

What's remarkable is that this requires no human labeling. Nobody had to tell the system that "cat" and "kitten" are related. The system figured it out from context alone.

This is called **self-supervised learning**: the training signal comes from the structure of the data itself, not from human annotations. It's why we can train on billions of words, since labeling that much data by hand would be impossible.

2.5.4 Static vs. Contextual Embeddings

There's an important distinction we should clarify. Word2Vec gives each word a single, fixed embedding regardless of context. The word "bank" gets the same embedding whether it appears in "river bank" or "bank account," a compromise position between its different meanings.

Modern LLMs improve on this significantly. They generate **contextual embeddings**: different embeddings for the same word depending on surrounding words. In a transformer model, "bank" gets one embedding in "river bank" and a completely different one in "bank account."

How? Through the attention mechanism we'll cover in Chapter 3. The initial word embedding is just a starting point; it gets modified based on context before being used.

So Word2Vec-style static embeddings were a crucial stepping stone, but modern LLMs have moved beyond them. Understanding static embeddings builds your intuition; understanding attention shows how we transcend their limitations.

2.6 From Words to Sentences

We've solved the problem of representing individual words. But language isn't just words, it's sentences, paragraphs, documents. How do we handle sequences?

2.6.1 The Naive Approach: Averaging

The simplest idea: average all the word embeddings in a sentence.

"The cat sat" → (embedding("the") + embedding("cat") + embedding("sat")) / 3

This gives you a single vector representing the sentence. And it works… sort of. For some tasks, like finding similar documents, averaging is surprisingly effective.

But there's a fatal flaw.

2.6.2 Order Matters!

Consider these sentences: - "Dog bites man" - "Man bites dog"

Same words. Same average embedding. Completely different meanings.

The first is unremarkable (dogs sometimes bite). The second is bizarre and newsworthy.

Or consider: - "The movie was not good" - "The movie was good, not bad"

Averaging would mix up "good" and "not" in similar ways for both sentences, missing that one is negative and one is positive.

2.6.3 The Problem

Word embeddings (at least static ones like Word2Vec) capture what words mean in isolation. But meaning in language depends on:

- **Word order**: "Dog bites man" "Man bites dog"
- **Long-range relationships**: "The cat that sat on the mat was…" What does "was" refer to?
- **Context modification**: "not good" changes the meaning of "good"
- **Disambiguation**: "I went to the bank" means different things near "river" vs. "money"

Averaging throws away all this information. We need something that can look at the full context and understand how words relate to each other.

2.6.4 What We Need

We need a way to:

1. Process words in sequence, maintaining order
2. Let words influence each other's meanings based on context
3. Handle relationships across long distances in text
4. Generate different representations for the same word in different contexts

This is exactly what the **attention mechanism** does, and it's the subject of Chapter 3. Attention allows each word to "look at" every other word and decide what's relevant. It's the innovation that transformed embeddings from a good idea into the foundation of modern AI.

For now, understand the limitation: static embeddings are necessary but not sufficient. They give us a mathematical starting point for words, but we need additional machinery to combine them into contextual meanings.

2.7 Hands-On Exercises

Theory is good. Experience is better. Let's see embeddings in action.

2.7.1 Online Tools to Explore

TensorFlow Embedding Projector (projector.tensorflow.org) - Visualizes word embeddings in 3D - You can search for words and see their neighbors - Try rotating the visualization to see different clusters

Word2Vec Demo Sites - Search for "word2vec online demo" - Many sites let you try word arithmetic - Type "king - man + woman" and see what comes out

2.7.2 Exercises

Exercise 1: Find Clusters

In an embedding visualizer, search for these words and note what clusters nearby: - "happy": What emotions cluster with it? - "doctor": What professions are neighbors? - "red": What other words are close?

Exercise 2: Try Word Arithmetic

If you find a word arithmetic demo, try:

- France - Paris + London = ?
- good - better + worse = ?
- dog - puppy + kitten = ?

Did you get the expected results? If not, why might that be?

Exercise 3: Find Odd Ones Out

Search for "apple" in an embedding visualizer. You might find two clusters:

1. Fruits (banana, orange, pear)
2. Technology (iPhone, Mac, Google)

The same word appears in different contexts and may have multiple positions (or a position that's a compromise between meanings).

2.8 Checkpoint Exercise

Time: 20-30 minutes **Materials**: Paper or spreadsheet (no code needed)

2.8.1 Instructions

Create a simple 3-dimensional embedding system for 10 animals.

1. **Choose 3 properties** that distinguish animals:
    - Example: size, domesticated, dangerous
    - Or: water-dwelling, number of legs, carnivore
2. **Rate each animal** from -1 to +1 on each property
3. **Create your embeddings**:

Animal	Size	Domestic	Dangerous
Cat	-0.5	0.9	-0.7
Lion	0.8	-1.0	0.9
Dog	0.0	0.9	-0.3
Goldfish	-0.9	0.8	-1.0
Shark	0.7	-1.0	0.9
Hamster	-0.9	0.9	-0.9
Bear	0.9	-0.9	0.8
Horse	0.7	0.7	-0.2
Wolf	0.3	-0.8	0.6
Rabbit	-0.7	0.7	-0.9

4. **Analyze your embeddings**:
    - Which animals are most similar? (closest in your 3D space)
    - Do "cat" and "lion" cluster together? (felines)

- Do "cat" and "hamster" cluster together? (small pets)
- Which grouping emerges more strongly?

5. **Reflection**:
   - Did your choice of dimensions determine what similarities emerge?
   - What dimensions would you add to better distinguish animals?
   - Can you see how 768 dimensions would capture more nuance?

2.9 Key Takeaways

What you learned:

1. ASCII codes capture spelling, not meaning ("cat" and "car" look similar but are unrelated)
2. Embeddings represent words as positions in a high-dimensional semantic space
3. Similar words cluster together; unrelated words are far apart
4. Word arithmetic works: king - man + woman ≈ queen
5. Context teaches meaning: words appearing together develop similar embeddings
6. Static embeddings have a limitation: averaging loses word order ("dog bites man" "man bites dog")

Key concepts:

- **Embedding**: A list of numbers (vector) representing a word's position in semantic space
- **Vector/Dimension**: More dimensions (768+) capture more nuanced relationships
- **Cosine similarity**: Measures how similar two embeddings are (1 = identical, 0 = unrelated, -1 = opposite)
- **Distributional hypothesis**: Words appearing in similar contexts have similar meanings ("You shall know a word by the company it keeps")
- **Self-supervised learning**: Learning from data structure, not human labels

The embedding pipeline:

```
Word: "king"
    ↓ embedding lookup
Vector: [0.82, 0.31, 0.91, -0.24, ...] (768 numbers)
    ↓ semantic space
Position near "queen", far from "banana"
```

> **Review Question Answers**
>
> All answers to these review questions are available in Appendix D.

2.10 Review Questions

1. Why can't we just use ASCII codes to represent words for AI?

2. What is an embedding, and why is it called that?
3. Why do modern embeddings use hundreds of dimensions rather than just 2 or 3?
4. Explain "king - man + woman = queen" in your own words. What does it reveal about embeddings?
5. What does "you shall know a word by the company it keeps" mean? How do embedding systems use this idea?
6. Why isn't averaging word embeddings enough for understanding sentences? Give an example.

2.11 What's Next

We now understand how individual words become numbers that capture meaning. But language is more than words in isolation, it's words in relationship.

Consider: "The cat sat on the mat because it was tired."

What does "it" refer to? The cat, obviously. But how would a computer know? The word "it" could refer to many things. Understanding requires looking at the whole sentence and figuring out what relates to what.

This is the problem of **attention**: how do we let each word "look at" other words to understand context? How do we process sequences while maintaining relationships?

That's Chapter 3: The Attention Mechanism. It's the key innovation that made modern LLMs possible.

Chapter 3

The Attention Mechanism

> "Concentrate every minute on doing what's in front of you with precise and genuine seriousness, tenderly, willingly, with justice. And on freeing yourself from all other distractions." — **Marcus Aurelius**, *Meditations*

What You'll Learn - Why earlier AI approaches struggled with long text - What "attention" means in the context of AI - How Query, Key, and Value work together - Why multi-head attention is more powerful than single attention - The 2017 paper that changed everything

Key Terms

- *Attention*: A mechanism that lets words selectively focus on other words, determining relevance and blending information accordingly See glossary
- *Self-Attention*: When a sequence attends to itself, where each word looks at all other words in the same sequence See glossary
- *Cross-Attention*: When one sequence attends to a different sequence, useful for translation, captioning, and other paired tasks See glossary
- *Query (Q), Key (K), Value (V)*: Three vectors derived from embeddings. Query represents what a word is looking for, Key represents what a word offers, and Value contains the actual information retrieved See glossary
- *Multi-Head Attention*: Running multiple attention calculations in parallel, each learning different relationship patterns See glossary
- *Softmax*: A function that converts scores into probabilities summing to 1; emphasizes relative differences so highest scores get most probability See glossary
- *Transformer*: The architecture from "Attention Is All You Need" (2017) that uses only attention mechanisms, no recurrence. Foundation of all modern LLMs See glossary
- *Scaled Dot-Product Attention*: Attention using dot products divided by $\sqrt{}$dimension, the standard approach in Transformers See glossary
- *Weights*: The learned numbers that define how transformations work. When we multiply an embedding by weights, we get Q, K, or V See glossary

- *Transformation*: The process of converting one set of numbers into another by multiplying by weights. Used to create Query, Key, and Value from embeddings See glossary

Checkpoint

By the end of this chapter, you'll understand:

1. How attention mechanisms work and why they revolutionized AI
2. How Query, Key, and Value enable words to find relevant context
3. How multi-head attention captures diverse relationships
4. Why the 2017 Transformer paper changed everything
5. The difference between self-attention and cross-attention

In Chapter 2, we learned how to turn words into numbers (embeddings that capture meaning). We also hit a wall: simple approaches like averaging embeddings lose crucial information. "Dog bites man" and "Man bites dog" have the same words, the same average embedding, but completely different meanings.

Word order matters. Context matters. Long-range relationships matter.

This chapter introduces the solution: the **attention mechanism**. It's the key innovation that made modern LLMs possible, and understanding it is essential for everything that follows.

3.1 The Problem with Old AI

Before attention, AI had a fundamental problem with language: it couldn't remember.

3.1.1 The Sequential Bottleneck

The dominant approach before 2017 used something called Recurrent Neural Networks (RNNs). Here's how they worked:

Imagine reading a sentence one word at a time, but you're only allowed to keep notes on a small sticky note. After reading each word, you update your sticky note, but the note stays the same size. By the time you reach the end of a long paragraph, the beginning is mostly forgotten, squeezed out by newer information.

```
Sentence: "The cat that the dog that the man owned chased ran away"

RNN processing:
Word 1: "The"     → [notes: "the"]
Word 2: "cat"     → [notes: "a cat"]
Word 3: "that"    → [notes: "cat that..."]
Word 4: "the"     → [notes: "cat that the..."]
Word 5: "dog"     → [notes: "cat... dog..."]
...
Word 10: "ran"    → [notes: "something... ran?"]
```

```
Word 11: "away"   → [notes: "ran away... wait, what ran?"]
```

By the time we reach "ran," we've lost track of what's doing the running. The cat? The dog? The man? The model has forgotten. In technical terms, this is called the **vanishing gradient problem**: information degrades as it travels through a chain.

3.1.2 What Long Text Requires

Consider this sentence:

"The **trophy** didn't fit in the **suitcase** because **it** was too big."

What does "it" refer to? The trophy (because the trophy is too big to fit) or the suitcase (because the suitcase is too big to… wait, that doesn't make sense)?

To answer this, you need to:

1. Remember both "trophy" and "suitcase" when you reach "it"
2. Use the meaning of "too big" to figure out which one makes sense
3. Connect words that are far apart in the sentence

Old AI struggled with all three. We needed something that could look back at any word, at any time, and understand what's relevant.

We needed attention.

3.2 What is Attention?

Attention solves the sequential bottleneck by letting every word "look at" every other word directly. No telephone game required.

3.2.1 The Cocktail Party

Imagine you're at a crowded cocktail party. Dozens of conversations surround you, a buzzing wall of sound. But somehow, you can focus on just one conversation. When someone across the room says your name, you immediately notice. Your brain is filtering the noise and attending to what matters.

That's attention: **selective focus based on relevance**.

Figure 3.1 illustrates this contrast. On the left, every voice competes equally (pure noise). On the right, your brain highlights what matters while dimming the rest. Notice the faded conversations aren't completely invisible. You're still somewhat aware of them, just paying less attention.

AI attention works similarly. Instead of processing words one by one through a chain, each word can directly examine every other word and decide what's relevant to its meaning.

3.2.2 From Sequential to Parallel

Here's the key shift:

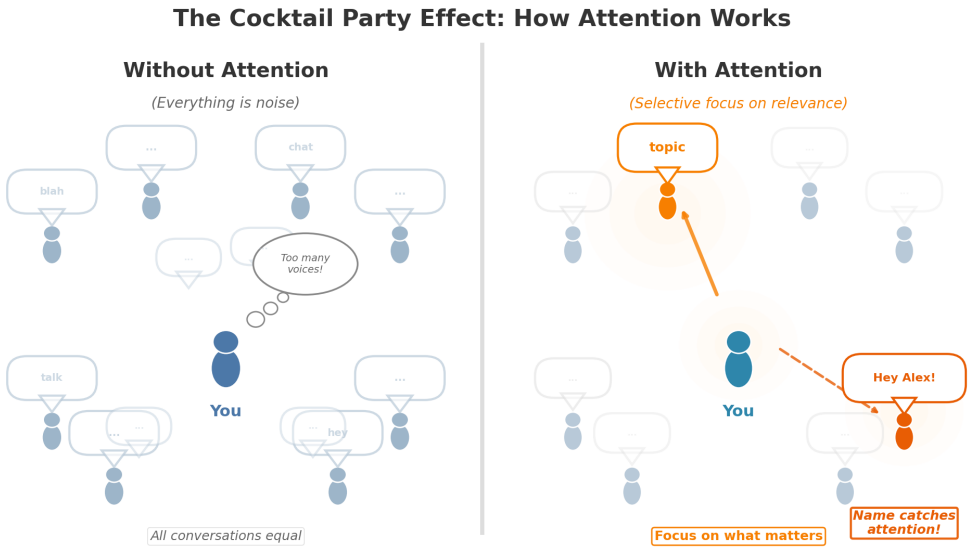

Figure 3.1: The Cocktail Party Effect: Without attention (left), all conversations compete equally, creating confusion. With attention (right), we selectively focus on relevant information, like hearing your name across a crowded room.

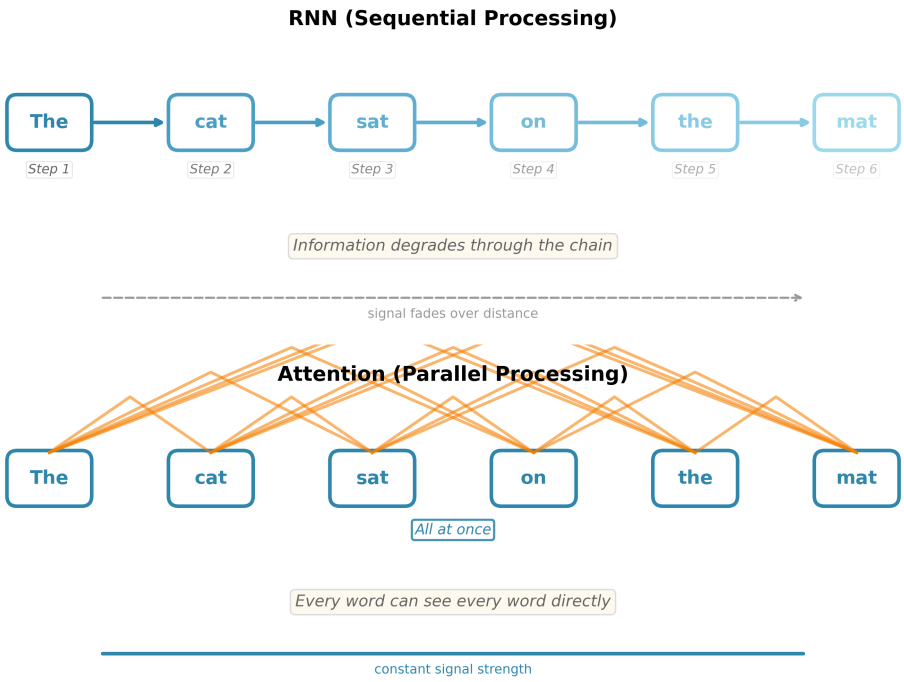

Figure 3.2: RNN processes words sequentially through a chain where information fades, while Attention lets every word directly access every other word.