AI and Large Language Models

From Turing to Transformers

Valerio Vaccaro

Satoshi Spritz Connect

March 24, 2026

Valerio Vaccaro

- Bitcoin Developer and Hardware Expert
- Contributor to Bitcoin open source projects
- DIY hardware enthusiast
- Bitcoin and Liquid Engineer at Blockstream

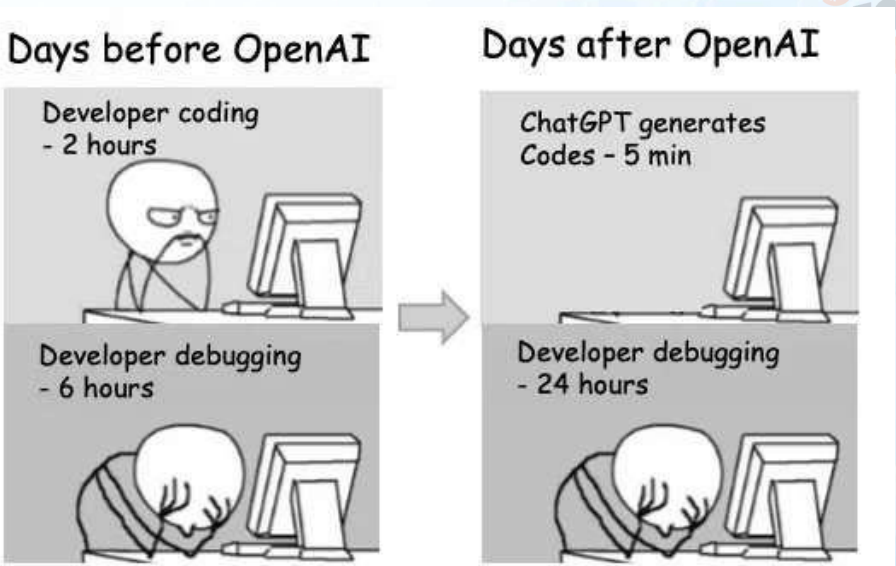
Social

- **LinkedIn** linkedin.com/in/valeriovaccaro
- **Github** github.com/valerio-vaccaro
- **Telegram** t.me/valeriovaccaro

License

May this presentation inspire you to become more self-sovereign!

Meme


Days before OpenAI
Developer coding
- 2 hours
Developer debugging
- 6 hours
Days after OpenAI
ChatGPT generates
Codes - 5 min
Developer debugging
- 24 hours

Outline

- 📜 Brief history of Artificial Intelligence
- 🤖 What are Large Language Models (LLMs)
- ⚙️ How LLMs work: tokens, embeddings, attention
- 💻 Build a minimal LLM in Python
- 📈 Benefits of LLMs vs other AI models

A Brief History of AI

The dawn (1940s–1960s)

- **1943** – McCulloch & Pitts: first mathematical model of a neuron
- **1950** – Alan Turing publishes *Computing Machinery and Intelligence* (Turing Test)
- **1956** – Dartmouth Conference: the term **Artificial Intelligence** is coined
- **1958** – Frank Rosenblatt builds the **Perceptron**, the first trainable neural network

AI history: winters and springs

Cycles of hype and disappointment

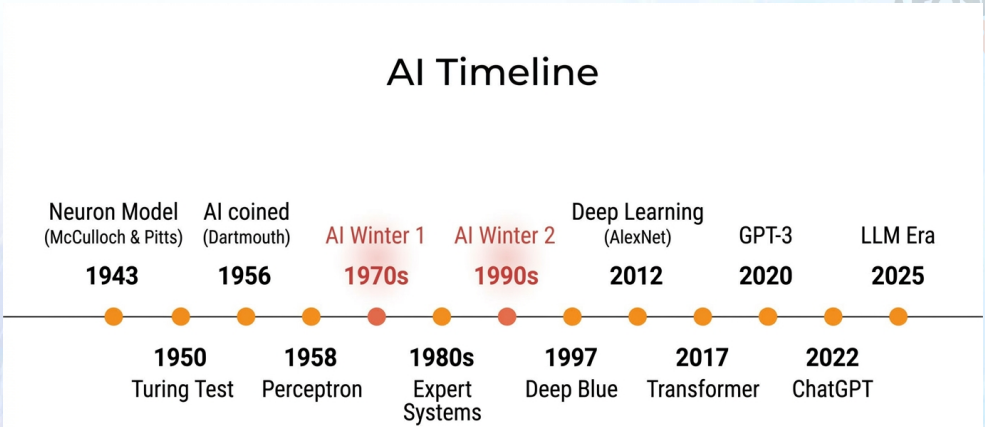
- 🔴 **1970s** – First **AI Winter**: limited compute, unmet promises
- 🚀 **1980s** – Expert Systems boom (rule-based AI)
- 🔴 **1990s** – Second AI Winter: expert systems fail to scale
- ✅ **1997** – IBM Deep Blue defeats Kasparov at chess
- 📈 **2012** – Deep Learning revolution: AlexNet wins ImageNet

AI history: the modern era

From deep learning to LLMs

- 🔥 **2014** – GANs (Generative Adversarial Networks) by Ian Goodfellow
- 🔑 **2017** – Google publishes *Attention Is All You Need* (Transformer architecture)
- 🤖 **2018** – OpenAI releases GPT-1 (117M parameters)
- 🚀 **2020** – GPT-3 (175B parameters): few-shot learning emerges
- 📈 **2022–2025** – ChatGPT, Claude, Llama, Gemini, DeepSeek: LLMs become mainstream

AI timeline


AI Timeline
Neuron Model
(McCulloch & Pitts)
1943
1950
Turing Test
AI coined
(Dartmouth)
1956
1958
Perceptron
AI Winter 1
1970s
1980s
Expert
Systems
AI Winter 2
1990s
1997
Deep Blue
Deep Learning
(AlexNet)
2012
2017
Transformer
GPT-3
2020
2022
ChatGPT
LLM Era
2025

What are LLMs?

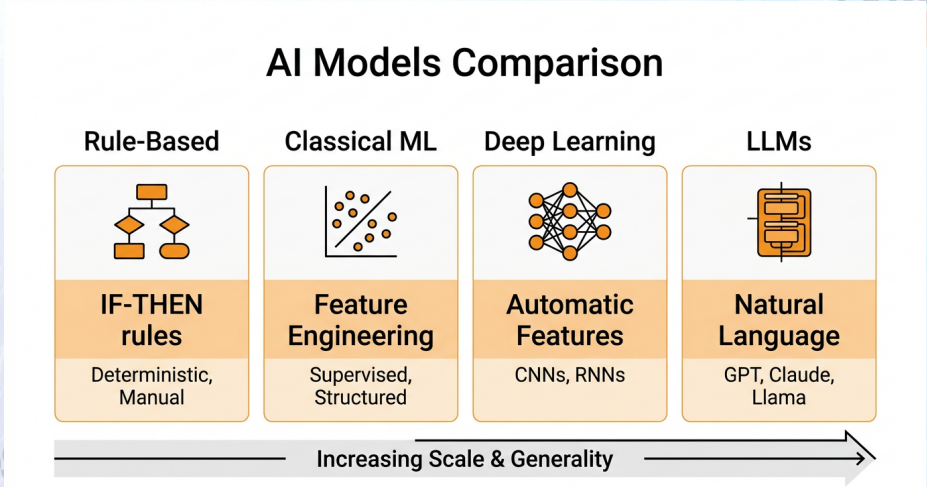
Large Language Models

- **Neural networks** trained on massive text corpora
- Goal: predict the **next token** in a sequence
- Learn patterns of language: grammar, facts, reasoning
- Scale matters: more parameters $\rightarrow$ more capable

Key insight

*An LLM is a **probability distribution over sequences of tokens**. Given a prompt, it generates text by repeatedly sampling the most likely next token.*

Types of AI models



How LLMs work: tokenization

From text to numbers

- ⚙ Text is split into **tokens** (words, subwords, or characters)
- 🔑 Each token maps to an integer ID via a **vocabulary**
- 📜 Example: `"Bitcoin is freedom"` → `[15496, 318, 7035]`

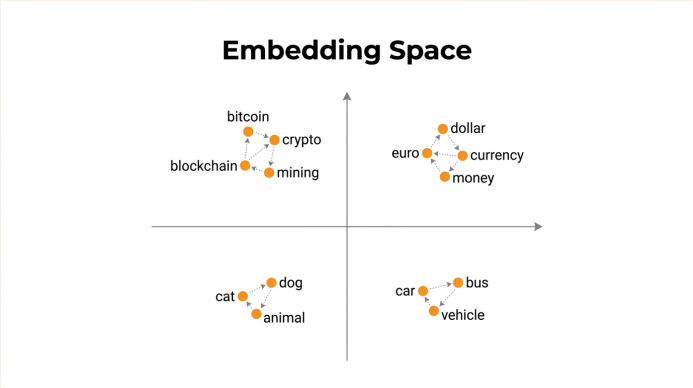
Common tokenizers

Tokenizer	Used by	Vocab size
BPE	GPT, LLaMA	50k–100k
WordPiece	BERT	30k
SentencePiece	T5, Gemma	32k–256k

How LLMs work: embeddings

Token to vector

- Each token ID is mapped to a **dense vector** (e.g. 768 or 4096 dimensions)
- **Positional encoding** adds information about token order
- Semantically similar words end up **close** in vector space



How LLMs work: the Transformer

Attention mechanism

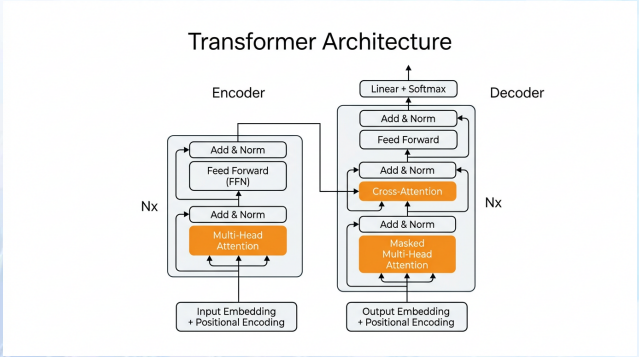
- **Self-attention**: each token attends to every other token
- Computes Query, Key, Value matrices from input
- Attention score: $\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right) V$

Transformer block

1. Multi-Head Self-Attention
2. Add & Normalize
3. Feed-Forward Network (MLP)
4. Add & Normalize

Stacked 12–96+ times depending on model size.

Transformer architecture



How LLMs generate text

Autoregressive generation

- ⚙ Given prompt tokens $[t_1, t_2, ..., t_n]$
- 🔑 Model predicts probability distribution over vocabulary for t_{n+1}
- 🔥 Sample or pick the most probable token
- 📜 Append t_{n+1} to the sequence, repeat

Sampling strategies

- **Greedy**: always pick highest probability
- **Temperature**: scale logits to control randomness
- **Top-k**: sample from top k most probable tokens
- **Top-p (nucleus)**: sample from smallest set with cumulative probability $\geq p$

Build a minimal LLM in Python

A character-level bigram model

The simplest language model: predict the next character based only on the current one.

```
import random

# Training corpus
corpus = "bitcoin is freedom and freedom is bitcoin"

# Build bigram transition table
bigrams = {}
for i in range(len(corpus) - 1):
    c1, c2 = corpus[i], corpus[i + 1]
    bigrams.setdefault(c1, []).append(c2)
```

Minimal LLM: generation

Sampling from the bigram model

```python
def generate(bigrams, start, length=60):
    result = start
    current = start
    for _ in range(length):
        if current not in bigrams:
            break
        nxt = random.choice(bigrams[current])
        result += nxt
        current = nxt
    return result

print(generate(bigrams, "b"))
# Example: "bitcon is freedom and freedom is bi..."
```

Minimal LLM: generation

Sampling from the bigram model

⚠ This is a **2-state** model (current char → next char). Real LLMs use the same principle but with billions of parameters and full context windows.

Minimal LLM: full code

Complete working example (20 lines)

```
import random

corpus = "bitcoin is freedom and freedom is bitcoin"

# Learn: count transitions
bigrams = {}
for i in range(len(corpus) - 1):
    c1, c2 = corpus[i], corpus[i + 1]
    bigrams.setdefault(c1, []).append(c2)
```

Minimal LLM: full code

Complete working example (20 lines)

```
# Generate: sample next character
def generate(bigrams, start, length=60):
    result, current = start, start
    for _ in range(length):
        if current not in bigrams:
            break
        current = random.choice(bigrams[current])
        result += current
    return result

for _ in range(3):
    print(generate(bigrams, "b"))
```

From bigram to Transformer

Scaling up the same idea

Feature	Bigram model	GPT-4 class
Context	1 character	128k+ tokens
Parameters	about 50	about 1.8 trillion
Training data	1 sentence	trillions of tokens
Architecture	Lookup table	Transformer stack
Generation	Random choice	Softmax + sampling

The **core idea is identical**: learn statistical patterns and predict the next element. The difference is scale and architecture.

Benefits of LLMs

Why LLMs outperform traditional AI

- ☑ **General purpose**: one model handles translation, coding, reasoning, summarization
- ☑ **Few-shot / zero-shot**: perform tasks with minimal examples
- ☑ **Natural interface**: interact using plain language
- ☑ **Transfer learning**: pre-train once, fine-tune for many tasks

LLMs vs traditional approaches

Rule-based systems

- 🔴 Require manual rule engineering
- 🔴 Brittle: fail on edge cases
- ✅ Deterministic and explainable

Classical ML (Random Forest, SVM, etc.)

- 🔴 Require feature engineering
- 🔴 One model per task
- ✅ Efficient on small structured data

LLMs

- ✅ No feature engineering needed
- ✅ Single model, many tasks
- ⚠️ Require large compute and data

LLMs vs Traditional AI

	Flexibility	Scalability	Explainability	Data needed
Rule-Based Systems	Flexibility: ★★☆☆☆ 1/5	Scalability: ★★☆☆☆ 1/5	Explainability: ★★★★★ 5/5	Data needed: Low
Classical ML (SVM, RF)	Flexibility: ★★★☆☆ 3/5	Scalability: ★★★☆☆ 3/5	Explainability: ★★★☆☆ 3/5	Data needed: Medium
LLMs (GPT, Claude)	Flexibility: ★★★★★ 5/5	Scalability: ★★★★★ 5/5	Explainability: ★☆☆☆☆ 1/5	Data needed: Very High

When NOT to use an LLM

LLMs are not always the best choice

- **Tabular/structured data**: XGBoost, Random Forest still win
- **Real-time low-latency**: LLM inference is slow
- **Deterministic logic**: use traditional algorithms
- **Small datasets**: classical ML is more data-efficient
- ⚠ **Hallucinations**: LLMs can generate plausible but false statements

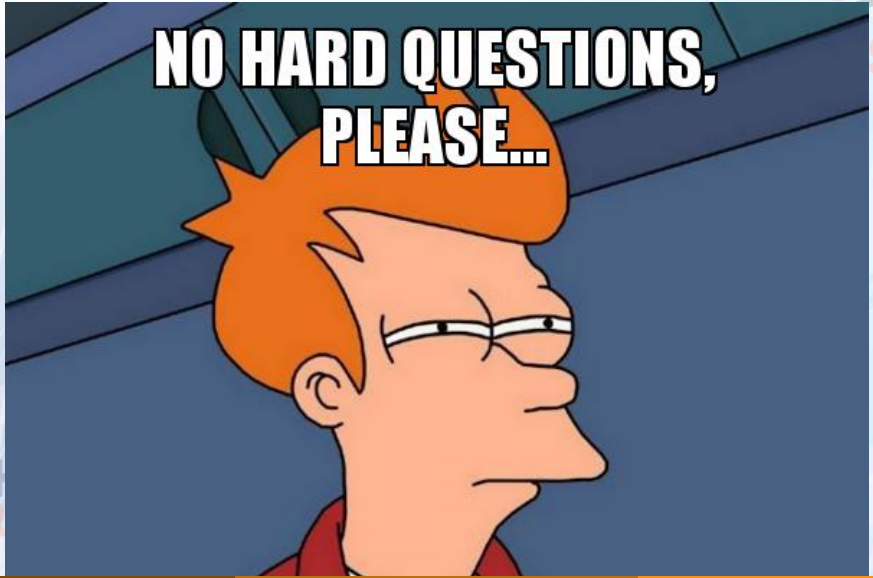
Rule of thumb

*Use an LLM when the task involves **understanding or generating natural language**.*
Use traditional models for everything else.

Conclusions

- AI has a long history of cycles; we're in a major **spring**
- LLMs are neural networks that predict the next token
- The Transformer architecture (attention) is the key breakthrough
- Even a 20-line bigram model captures the core principle
- LLMs excel at language tasks but aren't always the right tool
- Always evaluate: **LLM vs classical ML vs rule-based**

? Questions



References

- Vaswani et al., *Attention Is All You Need* (2017)
- Radford et al., *Language Models are Unsupervised Multitask Learners* (GPT-2, 2019)
- Brown et al., *Language Models are Few-Shot Learners* (GPT-3, 2020)
- Turing, *Computing Machinery and Intelligence* (1950)
- Andrej Karpathy, *Let's build GPT from scratch* (YouTube, 2023)

Online Resources

- arxiv.org/abs/1706.03762 – Transformer paper
- github.com/karpathy/nanoGPT – Minimal GPT implementation
- huggingface.co – Models and datasets

Satoshi Spritz

- 💻 Federation of local Bitcoiner groups
- 🎓 Free and privacy-oriented events
- 🤖 BITCOIN ONLY
- 🔧 Weekly online event: Satoshi Spritz Connect
- 📖 Focused on self-sovereignty education
- 🍕 Every week an online event → Satoshi Spritz Connect

Links

- satoshispritz.it
- t.me/SatoshiSpritzConnect

Officine Bitcoin

- 🤝 Italian Bitcoin Community, completely free
- 🤖 BITCOIN ONLY
- 🎓 Focus on education and project development
- 📋 Projects:
  - 💼 Bitcoin node development
  - 🔐 Hardware Wallet usage
  - 💻 Open source philosophy
  - 🤝 Debian installation
  - 🎲 Mnemonics & Dice
  - . . . and much more

Links

- officinebitcoin.it

Bitcoinissimo

- Bitcoin podcast and statistics
- Episodes in Italian, English, Hungarian, Chinese, Russian, Spanish, French
- Real-time network statistics, market data, block explorer

Links

- bitcoinissimo.it