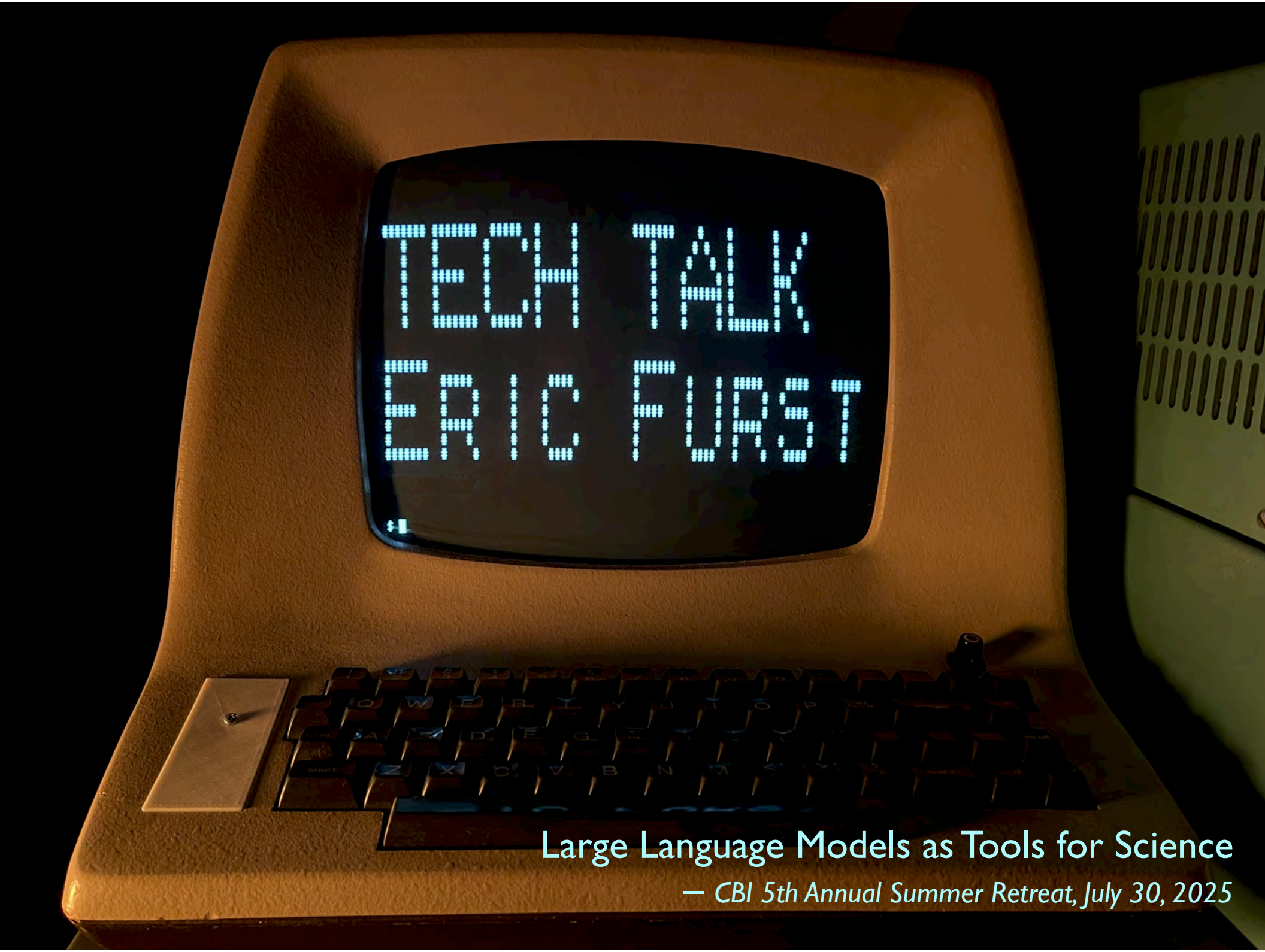
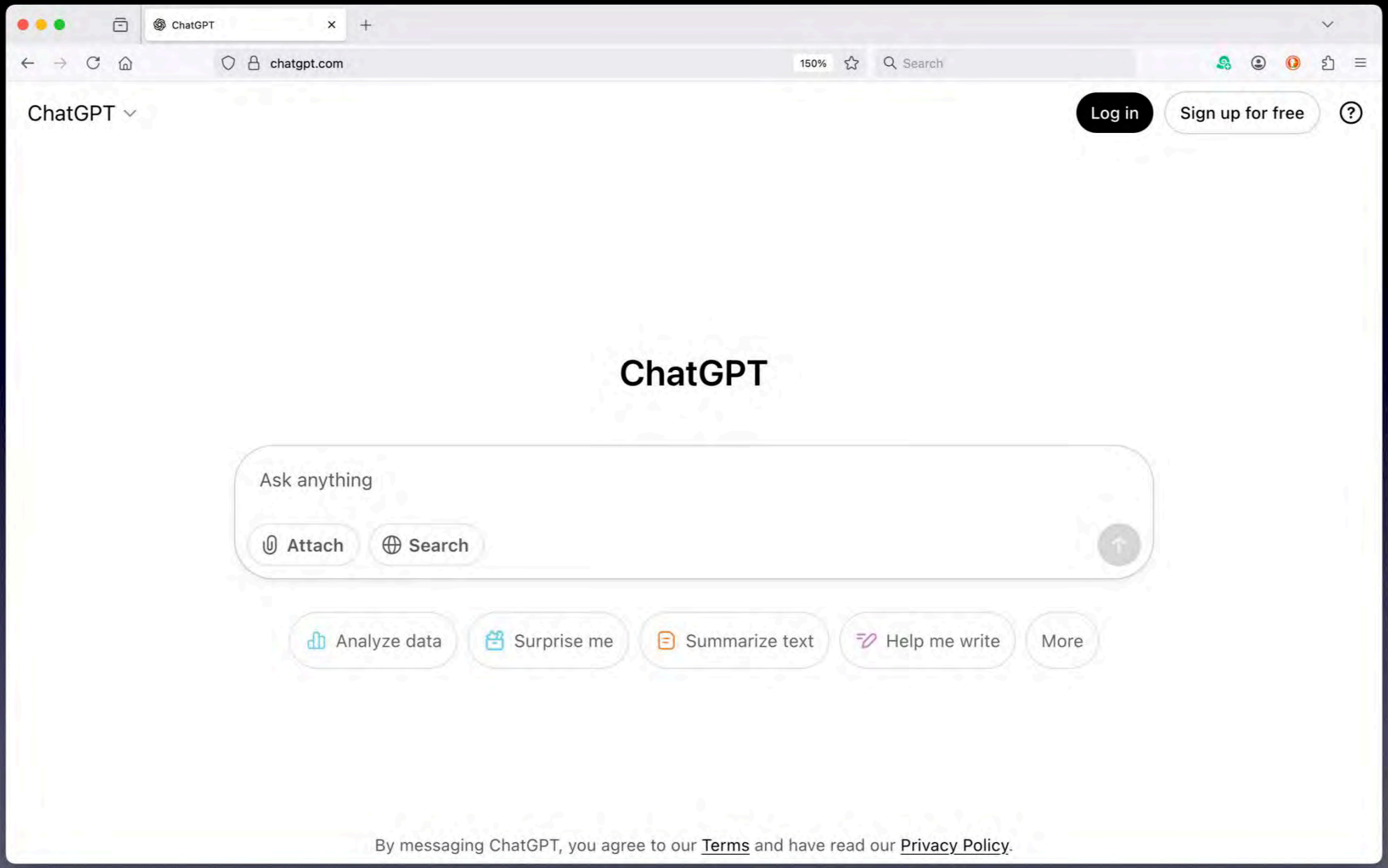


Large Language Models as Tools for Science

— CBI 5th Annual Summer Retreat, July 30, 2025

Outline

1. How LLMs work
2. Running LLMs locally
3. LLM uses


ChatGPT
chatgpt.com
150%
Search
ChatGPT
Log in
Sign up for free
ChatGPT
Ask anything
Attach
Search
Analyze data
Surprise me
Summarize text
Help me write
More
By messaging ChatGPT, you agree to our Terms and have read our Privacy Policy.

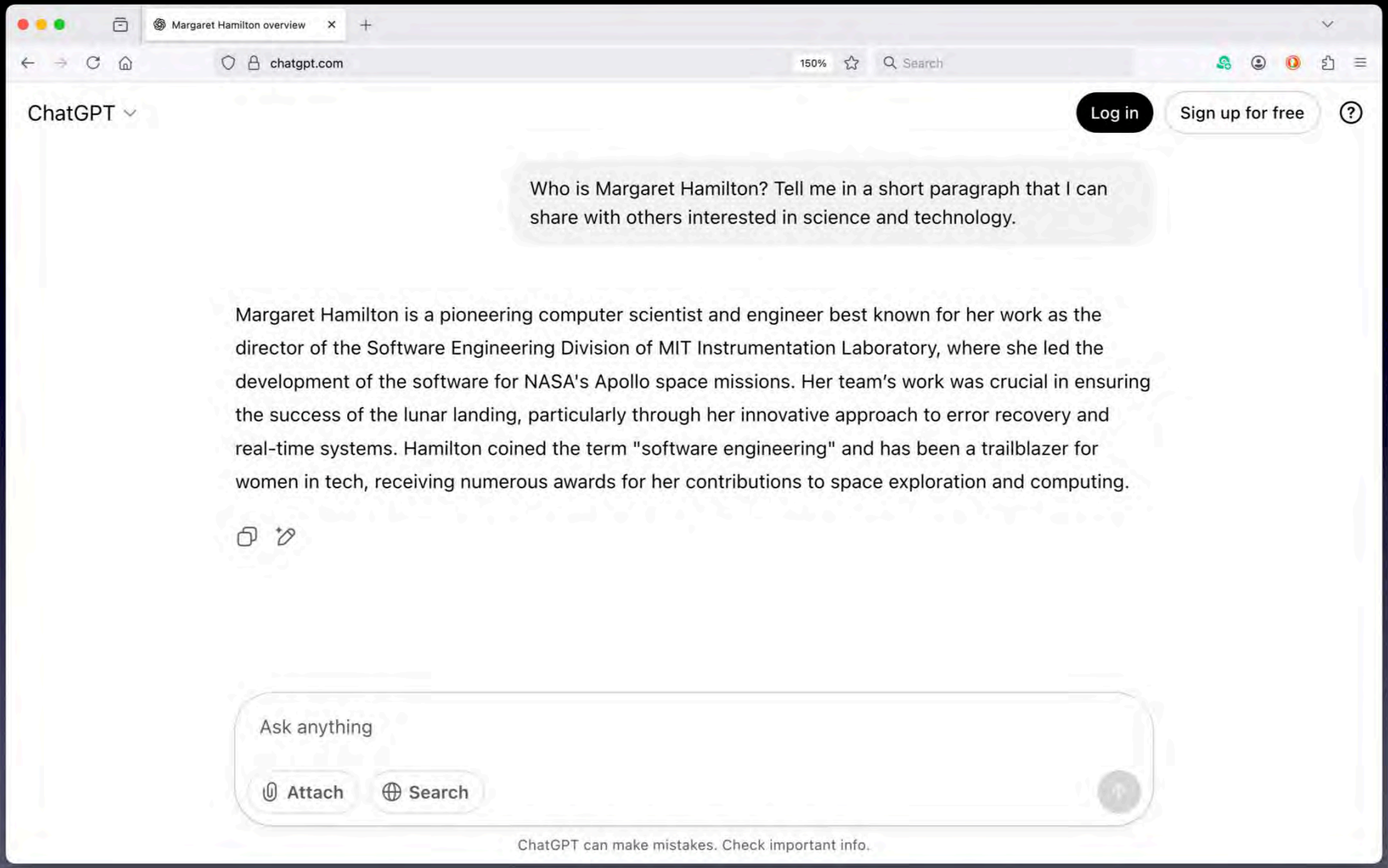
What happens?

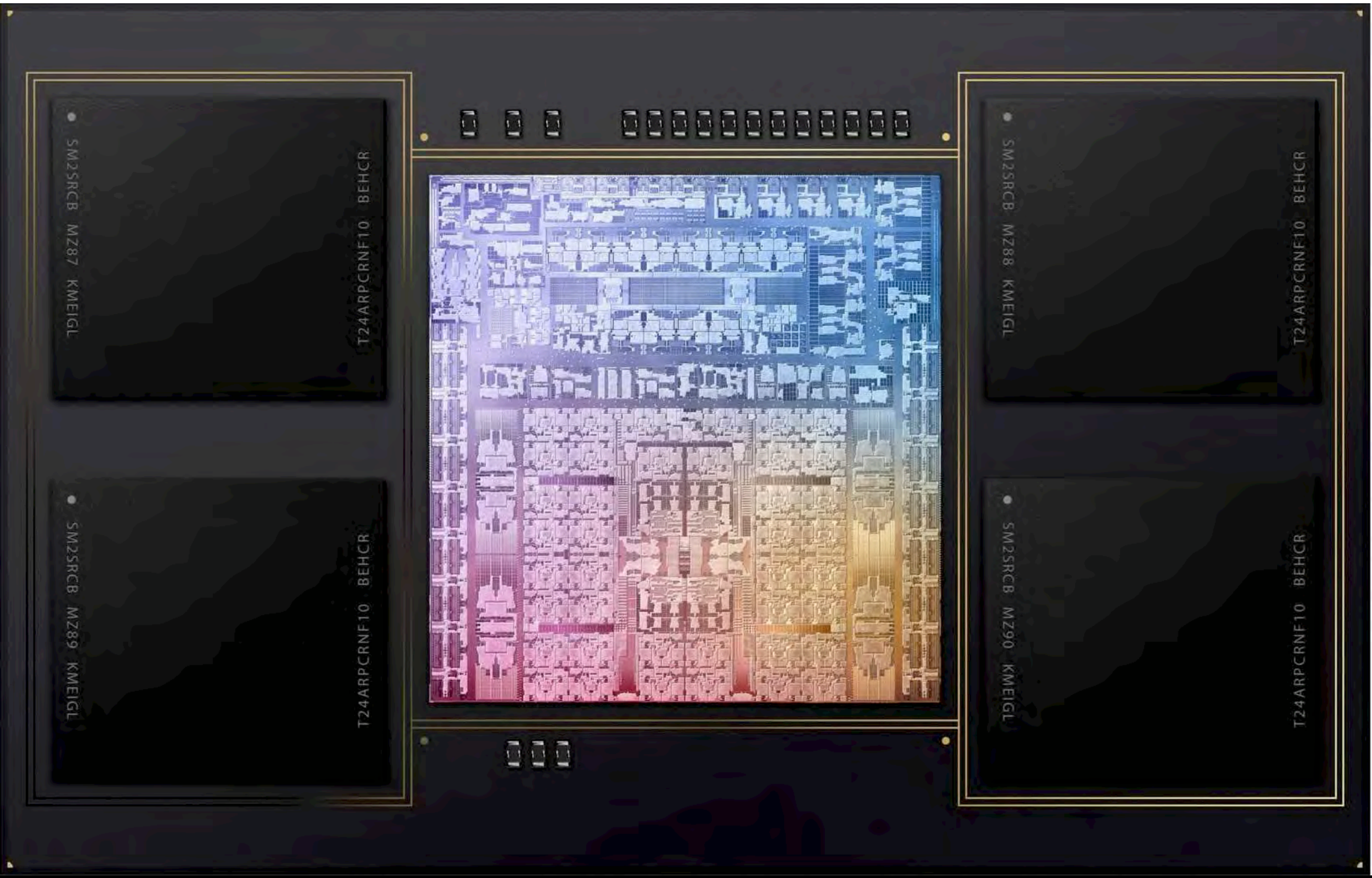
We input text — a "prompt"

The model responds with text

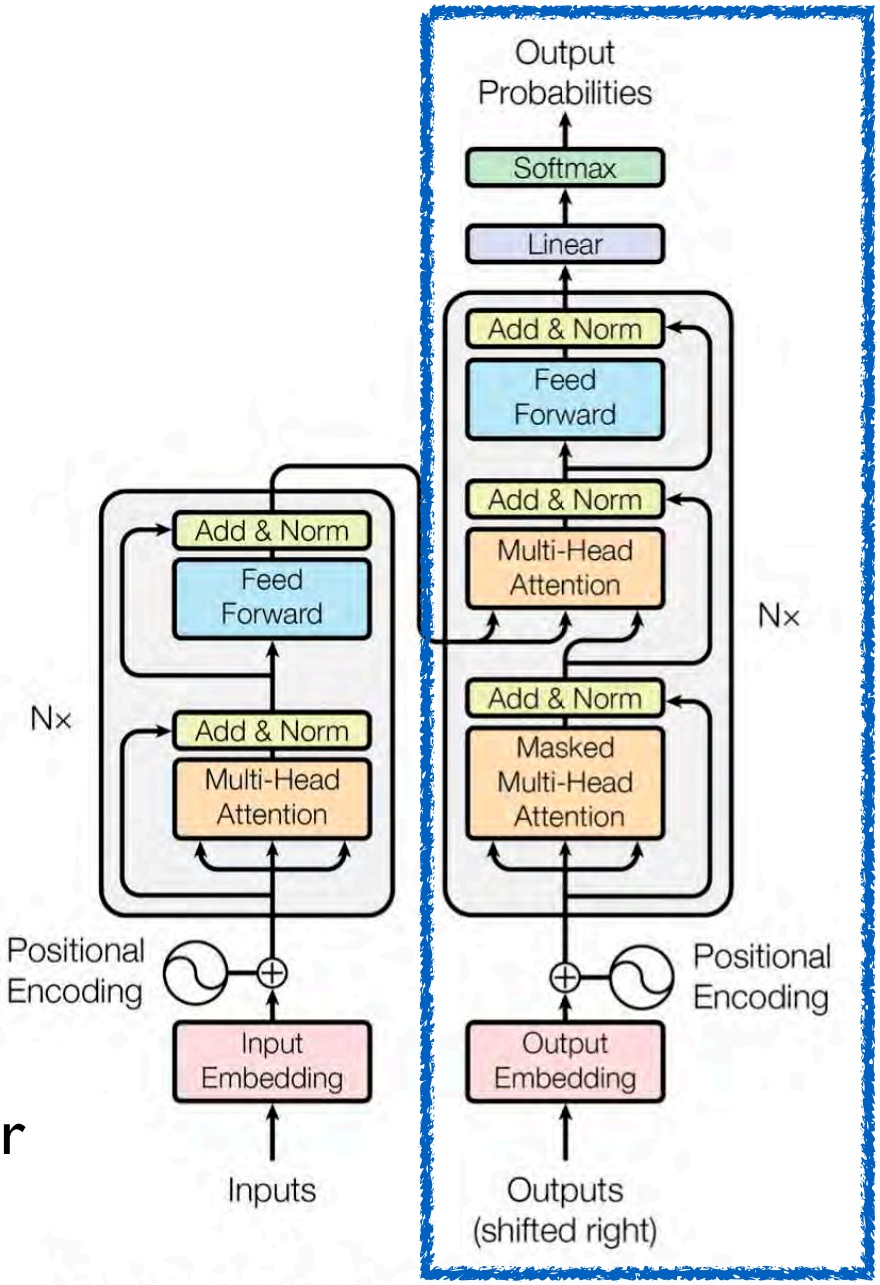
The quality of the response can depend strongly on the prompt

The response is never *exactly* the same


Margaret Hamilton overview
chatgpt.com
150%
Search
ChatGPT
Log in
Sign up for free
Who is Margaret Hamilton? Tell me in a short paragraph that I can share with others interested in science and technology.
Margaret Hamilton is a pioneering computer scientist and engineer best known for her work as the director of the Software Engineering Division of MIT Instrumentation Laboratory, where she led the development of the software for NASA's Apollo space missions. Her team's work was crucial in ensuring the success of the lunar landing, particularly through her innovative approach to error recovery and real-time systems. Hamilton coined the term "software engineering" and has been a trailblazer for women in tech, receiving numerous awards for her contributions to space exploration and computing.
Ask anything
Attach
Search
ChatGPT can make mistakes. Check important info.



GPU – Graphics Processing Unit



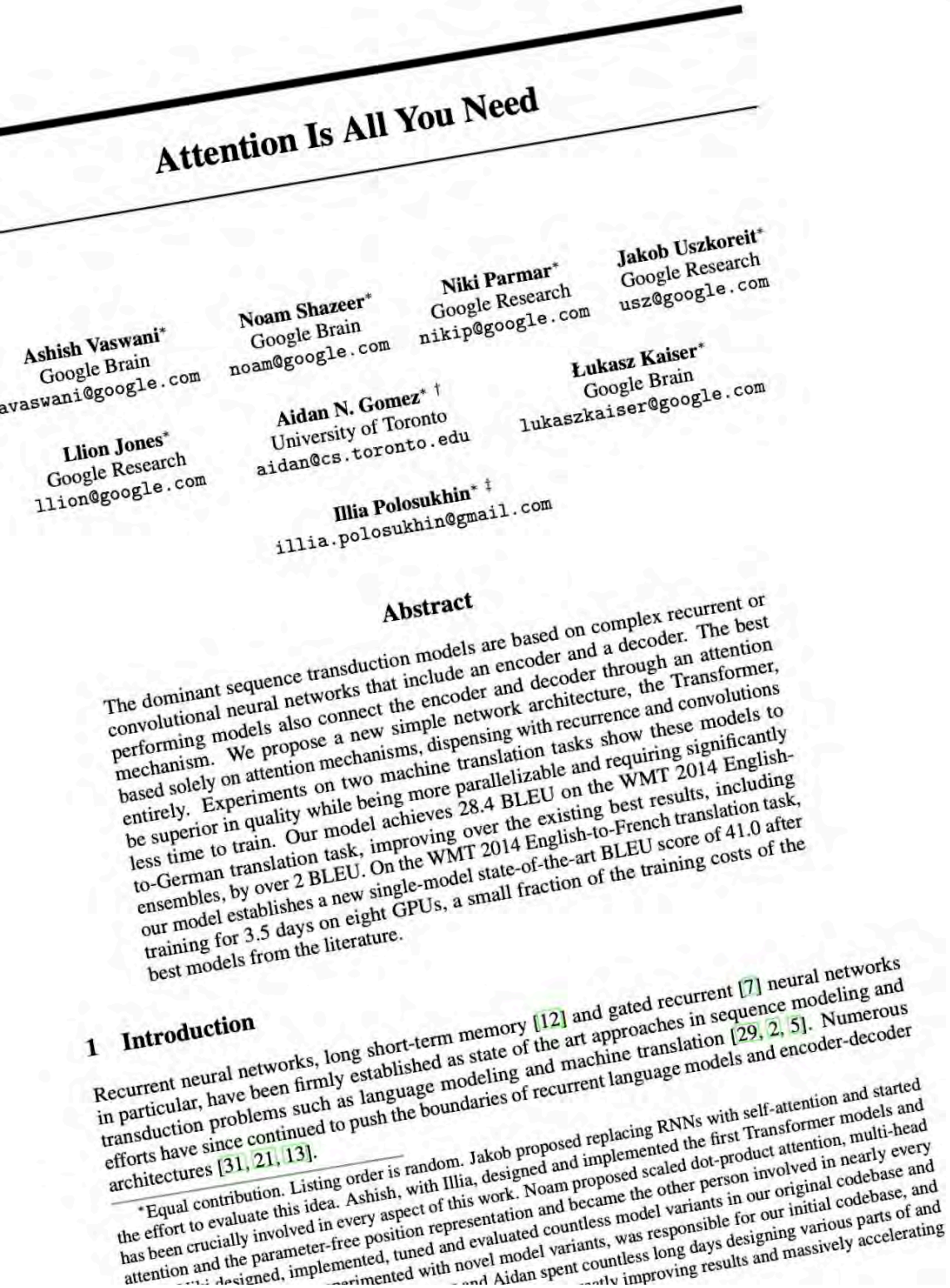
Encoder

Decoder

Figure 1: The Transformer - model architecture.

GPT = Generative Pre-trained Transformer

LLM is an auto-regressive language model that uses an optimized transformer architecture



Attention Is All You Need

Ashish Vaswani*
Google Brain
avaswani@google.com

Noam Shazeer*
Google Brain
noam@google.com

Niki Parmar*
Google Research
nikip@google.com

Jakob Uszkoreit*
Google Research
usz@google.com

Llion Jones*
Google Research
llion@google.com

Aidan N. Gomez* †
University of Toronto
aidan@cs.toronto.edu

Łukasz Kaiser*
Google Brain
lukaszkaiser@google.com

Illia Polosukhin* ‡
illia.polosukhin@gmail.com

Abstract

The dominant sequence transduction models are based on complex recurrent or convolutional neural networks that include an encoder and a decoder. The best performing models also connect the encoder and decoder through an attention mechanism. We propose a new simple network architecture, the Transformer, based solely on attention mechanisms, dispensing with recurrence and convolutions entirely. Experiments on two machine translation tasks show these models to be superior in quality while being more parallelizable and requiring significantly less time to train. Our model achieves 28.4 BLEU on the WMT 2014 English-to-German translation task, improving over the existing best results, including ensembles, by over 2 BLEU. On the WMT 2014 English-to-French translation task, our model establishes a new single-model state-of-the-art BLEU score of 41.0 after training for 3.5 days on eight GPUs, a small fraction of the training costs of the best models from the literature.

1 Introduction

Recurrent neural networks, long short-term memory [12] and gated recurrent [7] neural networks in particular, have been firmly established as state of the art approaches in sequence modeling and transduction problems such as language modeling and machine translation [29, 2, 5]. Numerous efforts have since continued to push the boundaries of recurrent language models and encoder-decoder architectures [31, 21, 13].

*Equal contribution. Listing order is random. Jakob proposed replacing RNNs with self-attention and started the effort to evaluate this idea. Ashish, with Illia, designed and implemented the first Transformer models and has been crucially involved in every aspect of this work. Noam proposed scaled dot-product attention, multi-head attention and the parameter-free position representation and became the other person involved in nearly every detail. Niki designed, implemented, tuned and evaluated countless model variants in our original codebase and tensor2tensor. Llion also experimented with novel model variants, was responsible for our initial codebase, and efficient inference and visualizations. Lukasz and Aidan spent countless long days designing various parts of and implementing tensor2tensor, replacing our earlier codebase, greatly improving results and massively accelerating

Attention Is All You Need, 2017

A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, Attention Is All You Need, in Advances in Neural Information Processing Systems, Vol. 30 (Curran Associates, Long Beach, CA, USA, 2017), pp. 261–272.

arXiv:1706.03762

followed by the…

First decoder-only GPT

P. J. Liu et al., "Generating Wikipedia by Summarizing Long Sequences," presented at the ICLR, 2018. Accessed: May 10, 2025. [Online]. Available: https://openreview.net/pdf?id=Hyg0vbWC-

Given a sequence of tokens (characters, words, bigrams, or subwords)

A logit is the raw…

Predict the next probable token

out…

Which becomes part of the next token prediction, etc.

(It is auto-regressive)

… put… of… a… ma… chine… learn ing mo del, typic ally be fore apply ing a trans form a tion like the soft max func tion.

Probabilistic but causal calculation of the next token

“Knowledge” is an emergent property

WIKIPEDIA
The Free Encyclopedia

Search Wikipedia | Search

Margaret Hamilton (software engineer)

50 languages

Article | Talk | Read | Edit | View history | Tools

From Wikipedia, the free encyclopedia

Contents hide

For other people named Margaret Hamilton, see Margaret Hamilton.

Margaret Elaine Hamilton (née **Heafield**; born August 17, 1936) is an American computer scientist. She was director of the Software Engineering Division of the MIT Instrumentation Laboratory, which developed on-board flight software for NASA's Apollo program. She later founded two software companies—Higher Order Software in 1976 and Hamilton Technologies in 1986, both in Cambridge, Massachusetts.

Hamilton has published more than 130 papers, proceedings, and reports, about sixty projects, and six major programs. She is one of the people credited with coining the term "software engineering".[1][2][3]

On November 22, 2016, Hamilton received the Presidential Medal of Freedom from president Barack Obama for her work leading to the development of on-board flight software for NASA's Apollo Moon missions.[4]

Early life and education [edit]

Margaret Elaine Heafield was born August 17, 1936, in Paoli, Indiana,[5] to Kenneth Heafield and Ruth Esther Heafield (née Partington).[6][7] The family later moved to Michigan,[8] where Margaret graduated from Hancock High School in 1954.[6]

She studied mathematics at the University of Michigan in 1955 before transferring to Earlham College, where her mother had been a student.[9][10] She earned a BA in mathematics with a minor in philosophy in 1958.[9][11] She cites Florence Long, the head of the math department at Earlham, as helping with her desire to pursue abstract mathematics and become a mathematics professor.[12]

Margaret Hamilton

Hamilton in 1995

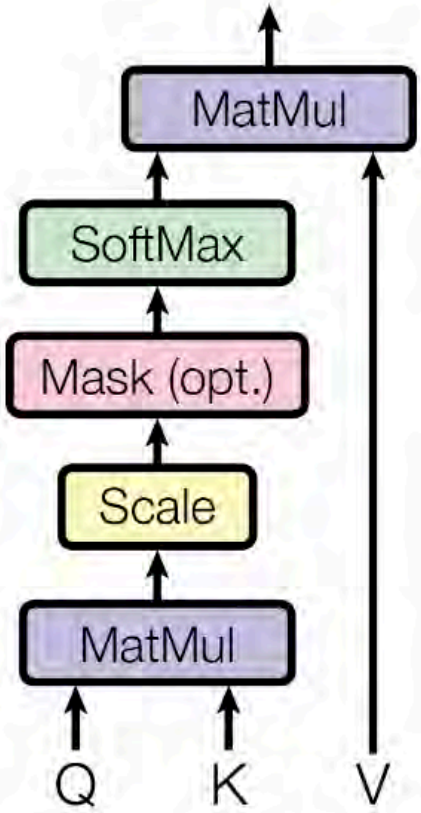
Born	Margaret Elaine Heafield August 17, 1936 (age 88) Paoli, Indiana, U.S.
Education	University of Michigan Earlham College (BA)
Occupation	Software engineer
Spouses	James Cox Hamilton (m. 1958; div. 1967) Dan Lickly (m. 1969)
Children	1
Relatives	James Cox Chambers (former son-in-law)

Example inspired by Andrej Karpathy, *Intro to Large Language Models*
https://www.youtube.com/watch?v=zjkBMFhNj_g

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V$$

$$\text{Mask} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 0 \\ 1 & 1 & 1 & 1 \end{bmatrix}$$

causality (for text)



Query, Key, and Value matrices

Probability of next token (SoftMax)

$$P_i = \frac{e^{z_i}}{\sum_{j=1}^{n} e^{z_j}}$$

Given vector of logits $z = [z_1, z_2, \ldots, z_n]$

Boltzmann weighting

$$P_i = \frac{e^{-E_i/k_B T}}{\sum_j e^{-E_j/k_B T}}$$

"Temperature" hyperscaling parameter

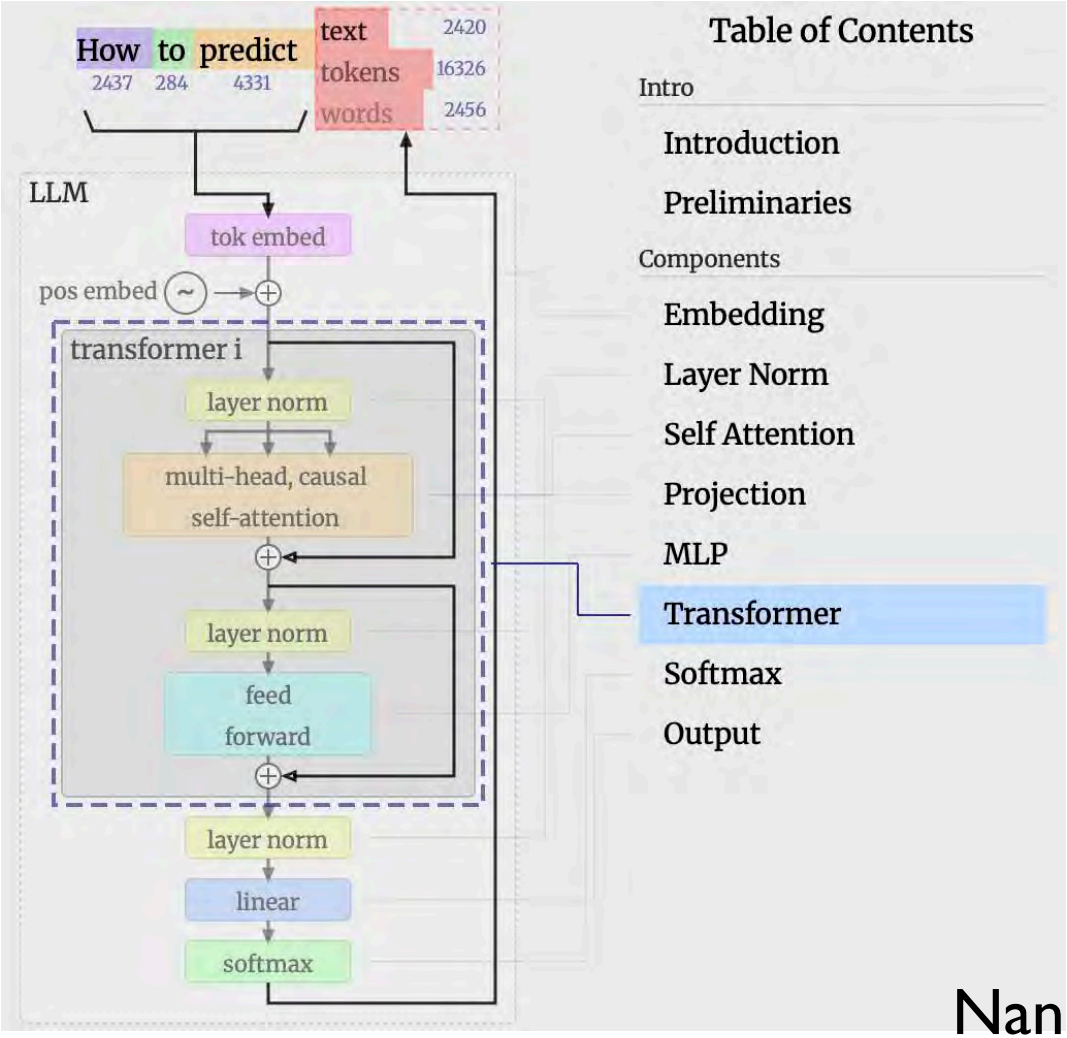
$$P_i = \frac{e^{z_i/T}}{\sum_{j=1}^{n} e^{z_j/T}}$$

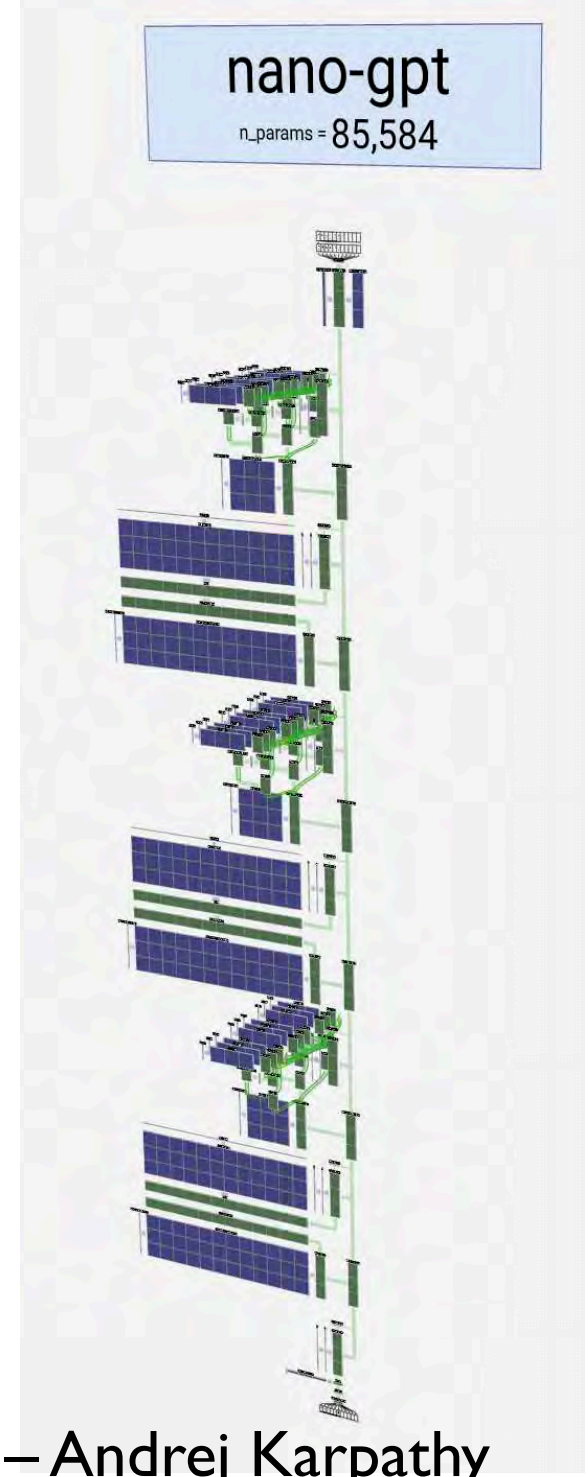
Stochastic behavior!

A. Vaswani, et al., Attention Is All You Need, in Advances in Neural Information Processing Systems, Vol. 30 (Curran Associates, Long Beach, CA, USA, 2017), pp. 261–272.

Visualizing a GPT

Brendan Bycroft – https://bbycroft.net/llm





NanoGPT – Andrej Karpathy

Given an input token representation, X

$$Q = XW_Q$$
$$K = XW_K$$
$$V = XW_V$$

Learnable weight matrices

each of size

$d \times d_k$ or $d \times d_v$ $(d \text{ is } n_{\text{embd}})$

the embedding dimension

GPT-3 (175B)

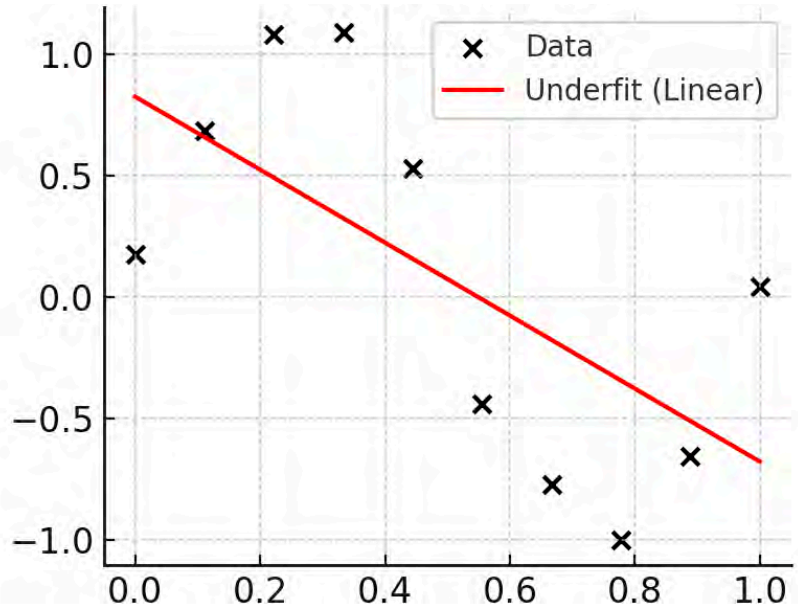
ca. June 2020

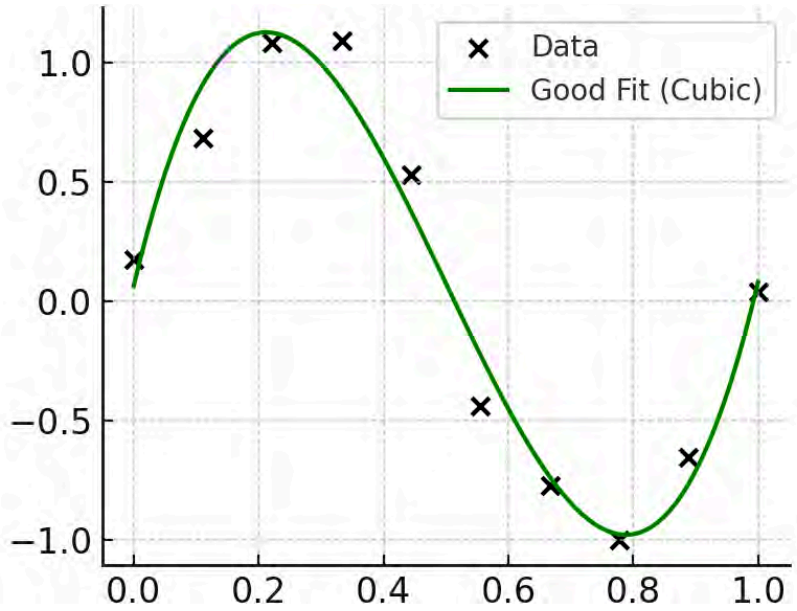
~175 billion parameters (350GB)
trained on *O*(10TB) data (the web)

- Hidden size: 12,288
- Number of layers: 96
- Number of attention heads: 96
- Vocabulary size: ~50,000
- Feedforward network expansion factor: 4x

GPT-4 and 4o sizes not released, but estimates at 10^{12} – 10^{14} parameters

Training model weights





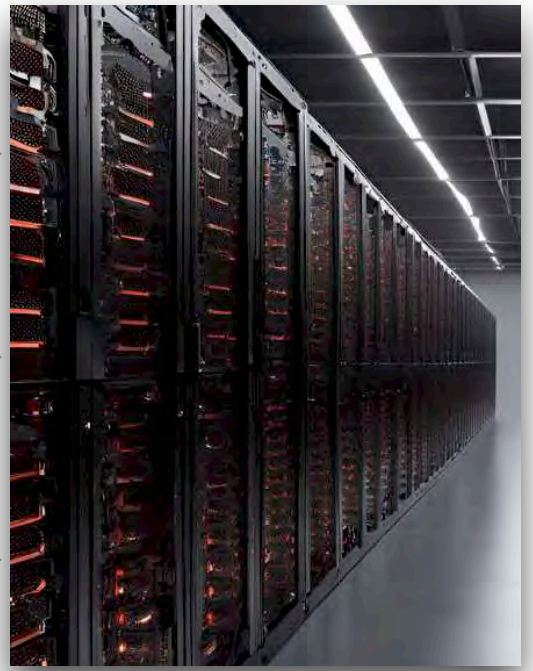
Parameters (model weights) adjusted to better fit the model
Instead of *sum of squared residuals*, use *cross-entropy loss*

Cost (compute & energy) is in the model training

~10 TB text

webcrawl, Wikipedia, Project Gutenberg, ArXiv, Stack Exchange… (llama)

+ fine-tuning



Llama 3.1 (Meta)

8B model: 4.9 GB
70B model: 43 GB
405B model: 243 GB

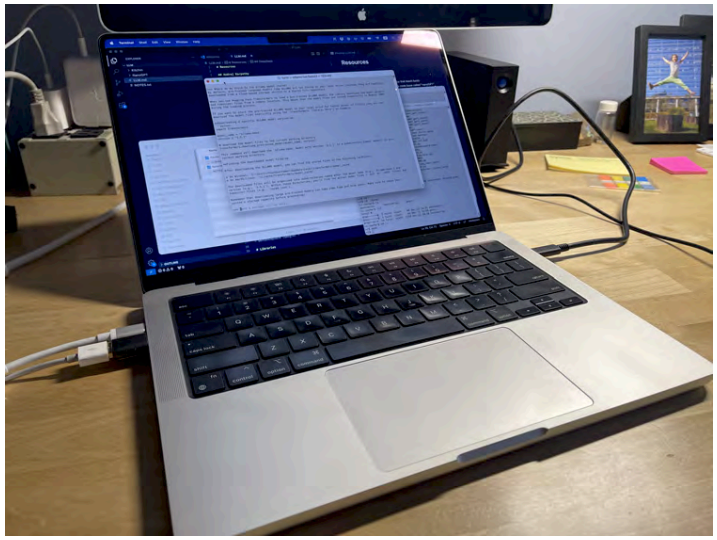
Llama 3.1 8B –1.46 million GPU hours
Llama 3.1 70B – 7.0 million GPU hours
Llama 3.1 405B – 30.84 million GPU hours

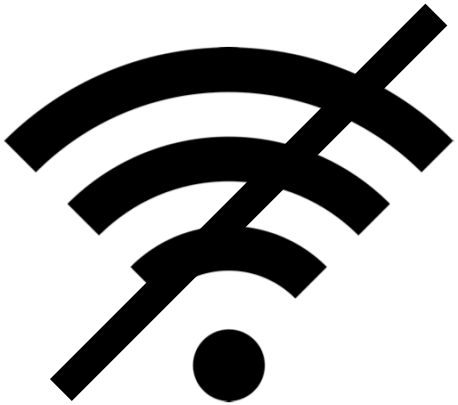
(approximately 440,000 petaFLOP-days for training)

https://huggingface.co/blog/llama31
https://ollama.com/library/llama3.1

Running models is relatively *inexpensive*

Macbook Pro M3, 12 cores (2023)
18GB RAM, 18 core GPU





running in *ollama* locally
with llama3 8B (4.7 GB)

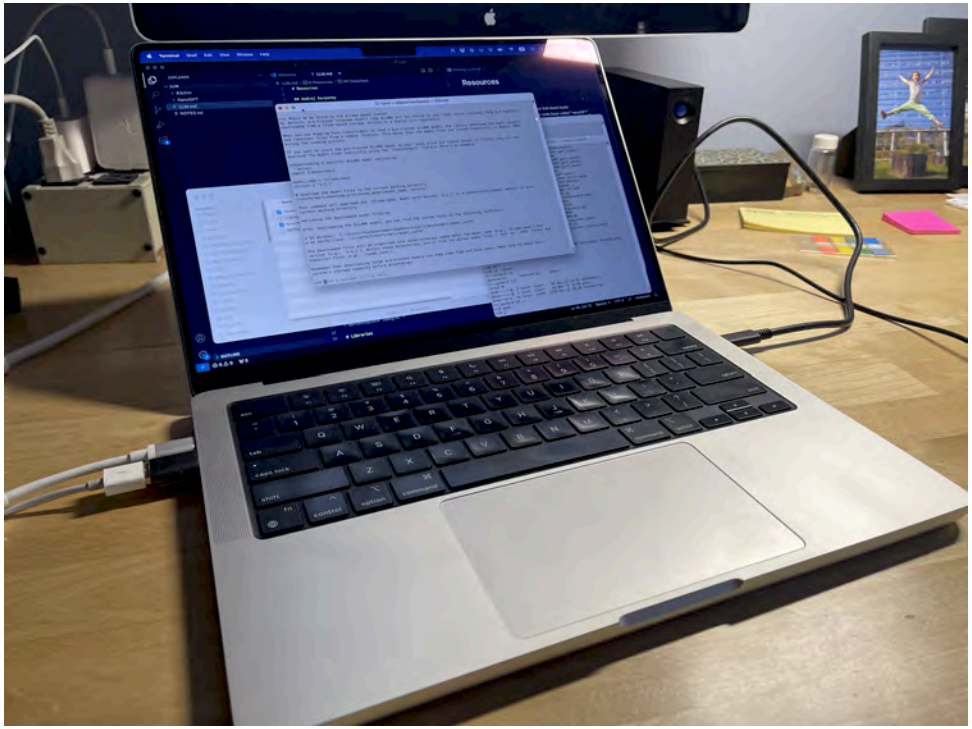
```
-rwxr-xr-x@ 1 furst  staff    51M Jan 16 11:37 ollama*
```

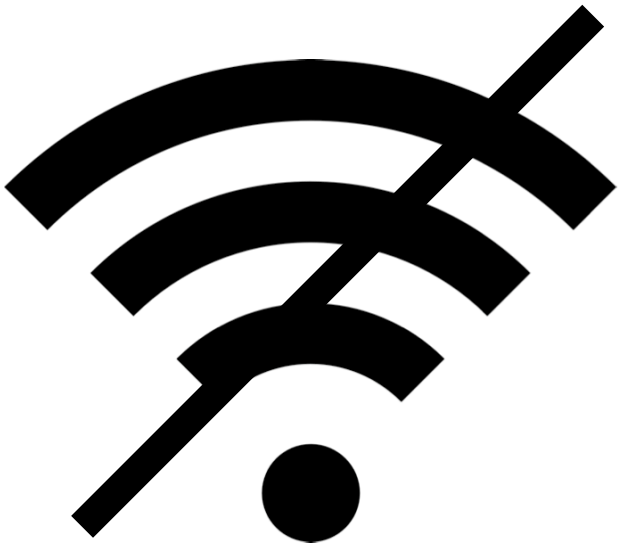
HTTPS://OLLAMA.COM/

furst — ollama run llama3 — 122×40

```
~/$ ollama run llama3
>>> Say hello to the Winter Research Review.
A warm welcome to the Winter Research Review!

As the snowflakes gently fall outside, we're excited to bring you the latest and greatest in research discoveries
from various fields. Whether you're a scientist, student, or simply someone curious about the world around us, this
review is designed to provide a comprehensive overview of the most significant findings from the past year.

From groundbreaking advancements in medicine and technology to intriguing insights into human behavior and the
natural world, we've got you covered. So, grab a hot cup of cocoa, get cozy, and let's dive into the Winter Research
Review!

>>> How do I finetune ollama models on my own text?
Finetuning OLLAMA (Open Large-scale LAnguage Model Architecture) models on your own text can be a fantastic way to
adapt the model to your specific domain or use case. Here's a step-by-step guide to help you finetune an OLLAMA
model:

**Prerequisites:**

1. **Text dataset**: Prepare a large text dataset related to your topic of interest. This dataset will serve as the
training data for fine-tuning the model.
2. **OLLAMA pre-trained model**: You can use any publicly available OLLAMA pre-trained model, such as those provided
by Hugging Face Transformers or Google's BERT models.
3. **Python and necessary libraries**: Install Python (3.x) and the required libraries:
        * `transformers` for working with pre-trained language models
        * `torch` for building and training neural networks

**Step-by-Step Guide:**

1. **Prepare your text dataset**:
        * Preprocess your text data by tokenizing, normalizing, and encoding it using a suitable tokenizer (e.g.,
`bert-base-uncased`).
        * Split your dataset into training (~80%), validation (~10%), and testing sets (~10%).
2. **Load the pre-trained OLLAMA model**:
        * Import the required libraries: `transformers`, `torch`
        * Load the pre-trained OLLAMA model using the `AutoModelForSequenceClassification` class from Hugging Face
Transformers
3. **Create a finetuning dataset class**:
        * Define a custom dataset class that inherits from `Dataset` (or `torch.utils.data.Dataset`)
        * Implement the `__getitem__` and `__len__` methods to handle the text data and its corresponding labels (if any)
```

Demo: running models locally





Macbook Pro M3, 12 cores (2023)
18GB RAM, 18 core GPU

llama3 8B (4.7 GB)
running in *ollama*

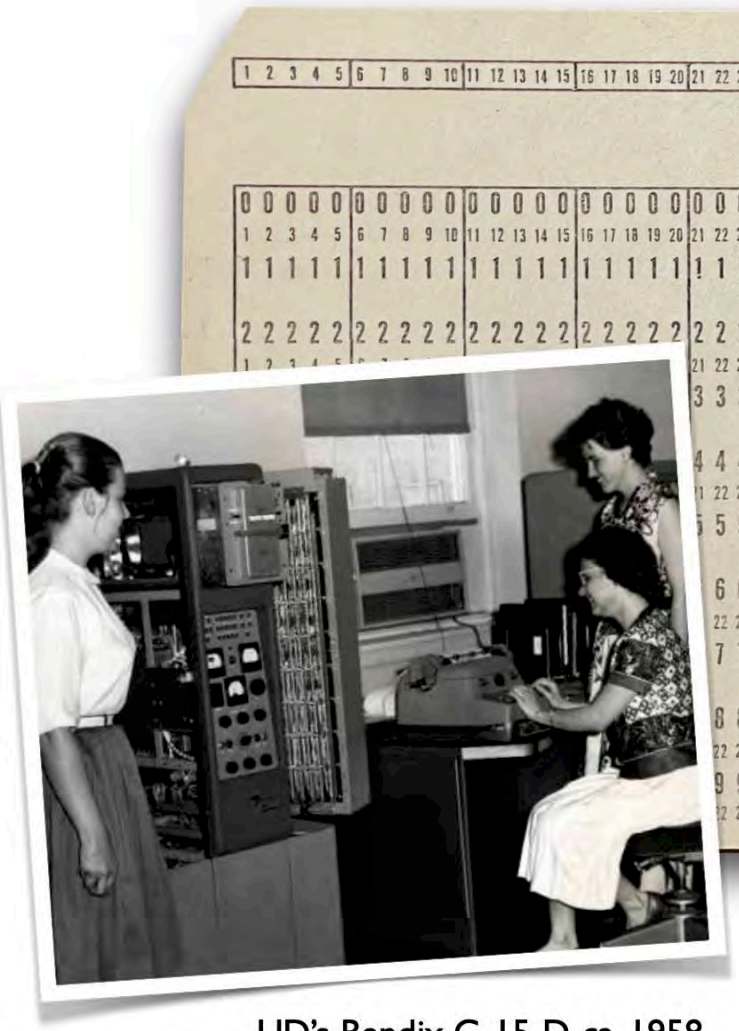
HTTPS://OLLAMA.COM/

Outline

1. How LLMs work
2. Running LLMs locally
3. LLM uses

LLM uses in science and engineering

GPTs and LLMs are *transformative technologies*, analogous to the advent of the electronic, programmable digital computer

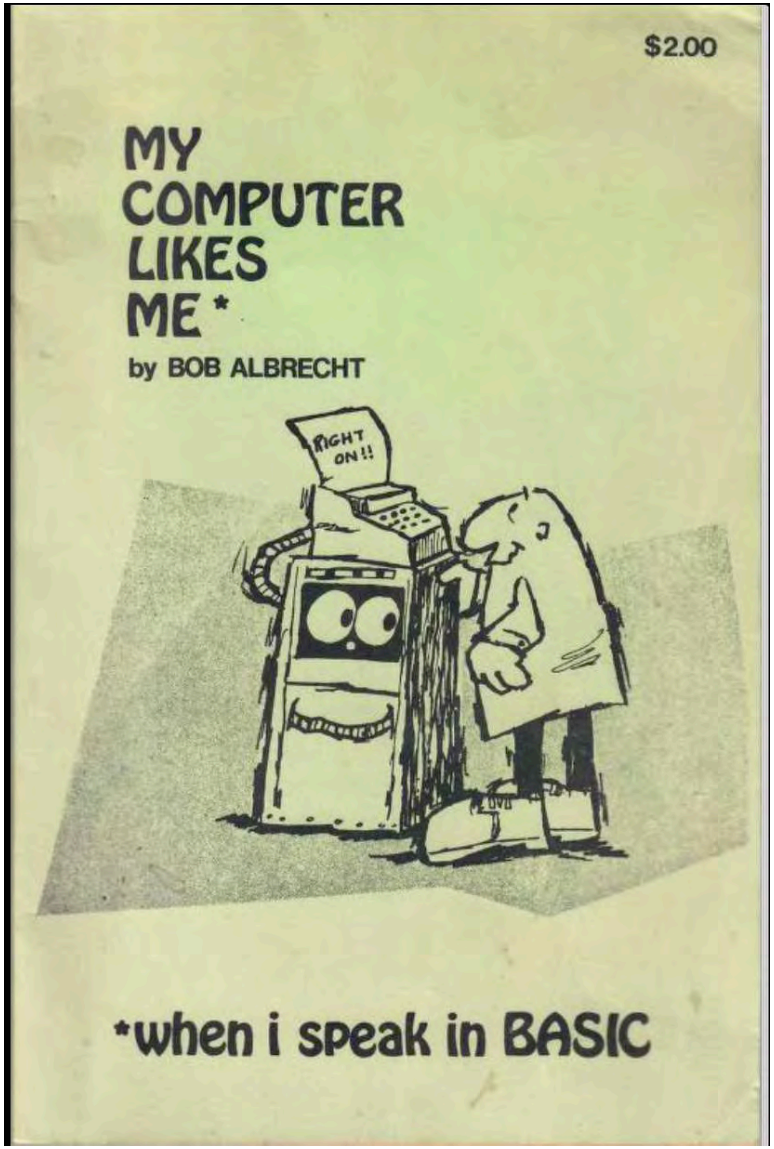


UD's Bendix G-15-D, ca. 1958

Scientists and engineers rapidly adopted digital computers to numerically solve difficult problems

Blaszczyk, 2014

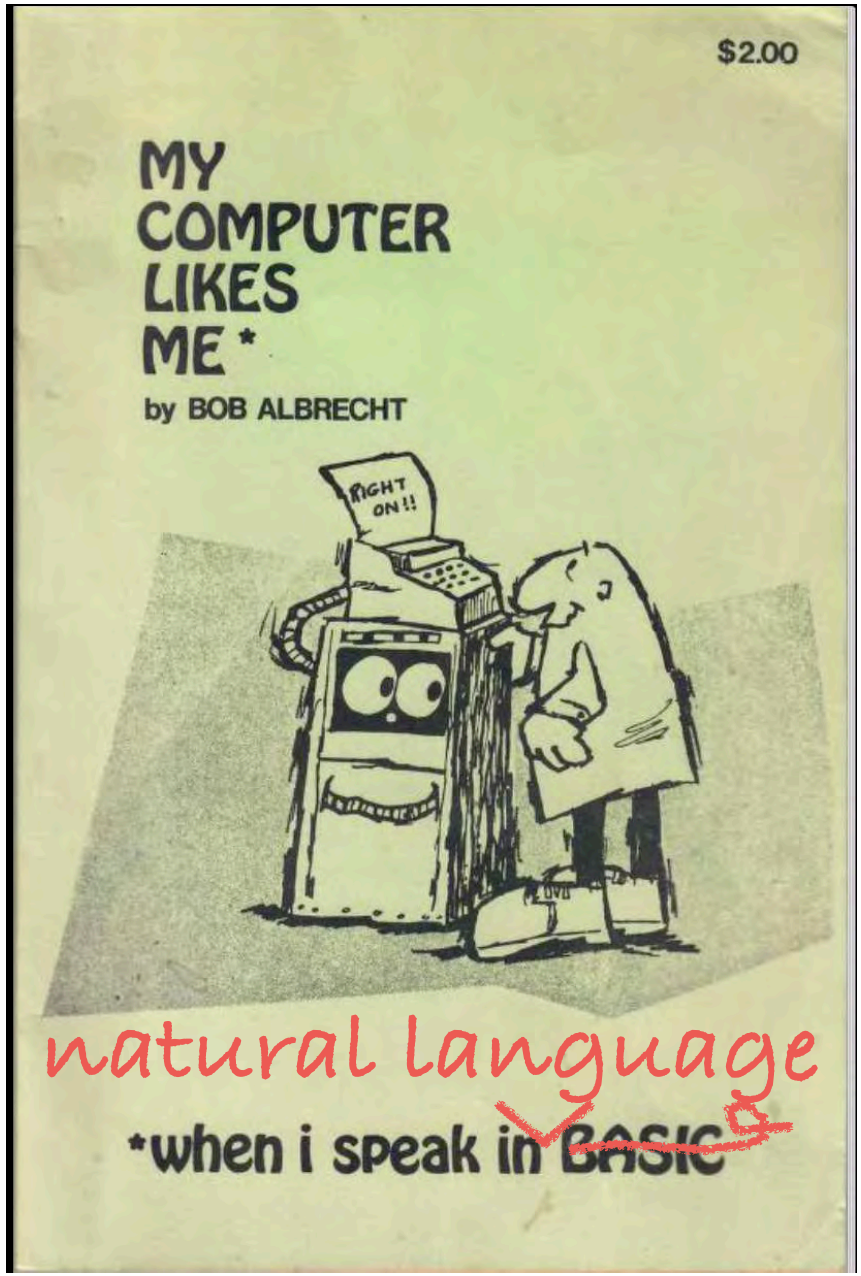
Since 1972 (and earlier)…



or…

assembly
FORTRAN
C
Matlab
Python
Javascript, Pascal, Lisp, ALGOL 60, BAL, JCL, Smalltalk, PL/I, Logo…

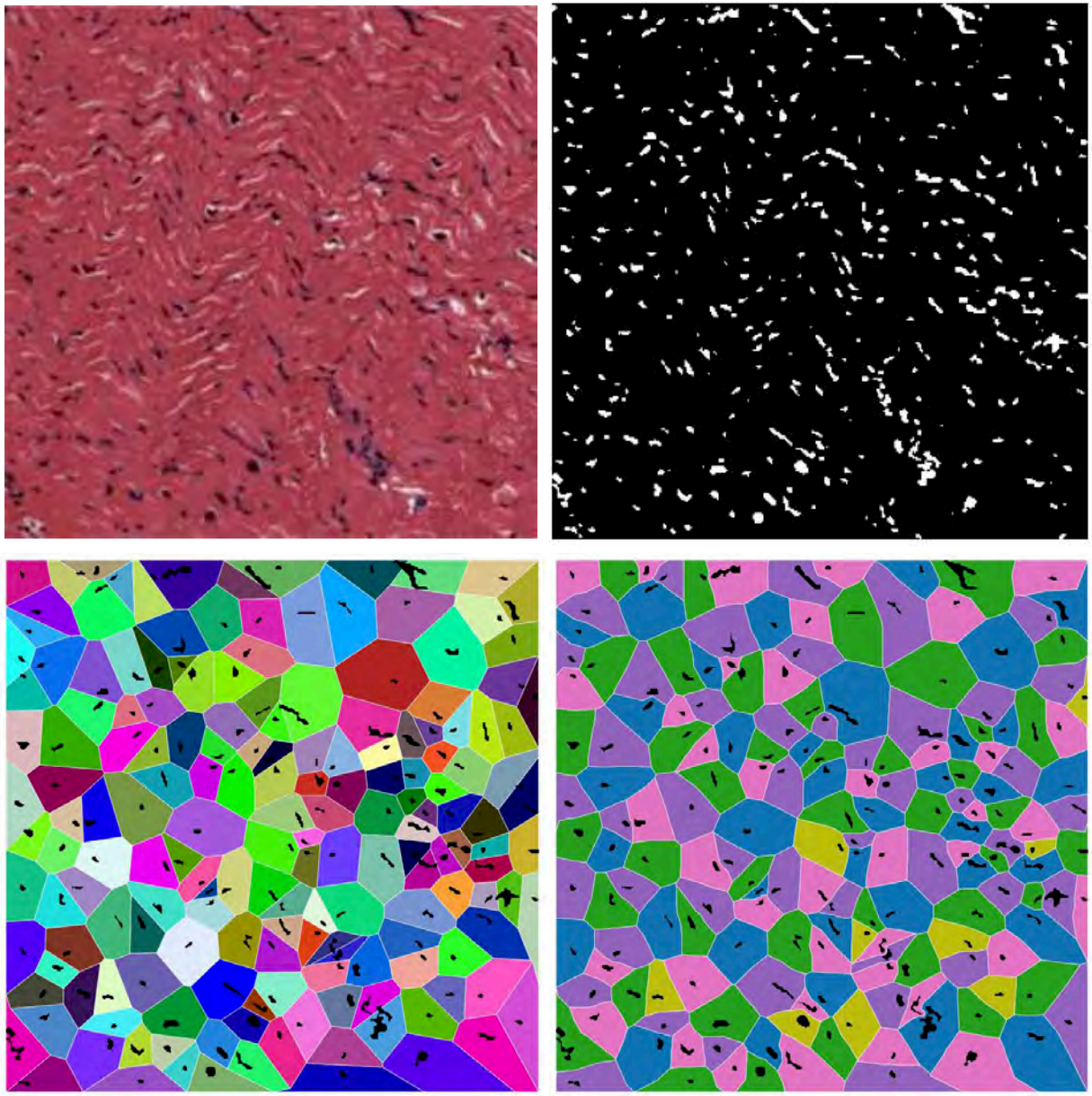
suddenly,
today…



Solving non-trivial, but adjacent problems

Jason Conradt

Minimum color Voronoi representation



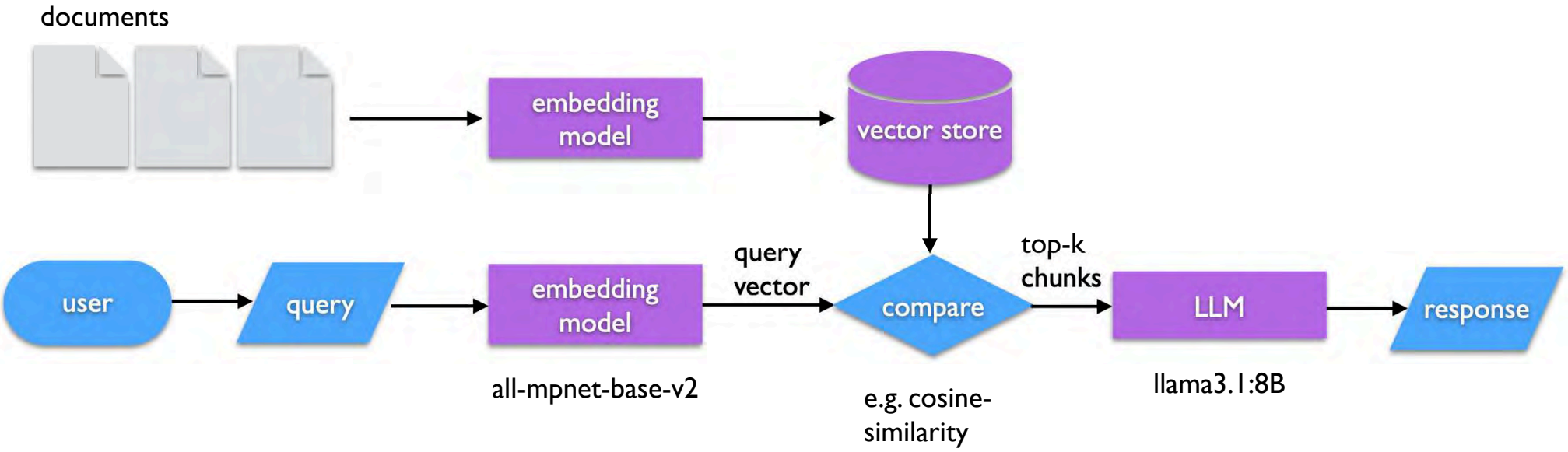
Use ChatGPT to help write Python code for binarization and Voronoi tessellation… in a day instead of a week +

Retrieval Augmented Generation (RAG)

Pull in unstructured text, PDFs, documents, webpages and more and index the data within them

The simplest queries involve either semantic search or summarization –

- Semantic search: Query about specific information in document(s) that matches the query terms and/or semantic intent
- Summarization: condensing a large amount of data into a short summary relevant to your current question



semanticsearch_local — furst@anisotropic: ~ — emacs -nw query.py — 129×49

```
File Edit Options Buffers Tools Python Help
# query.py
# Run a querry on a vector store
#
# E.M.F. July 2025

from llama_index.core import (
    StorageContext,
    load_index_from_storage,
    ServiceContext,
    Settings,
)
from llama_index.embeddings.huggingface import HuggingFaceEmbedding
from llama_index.llms.ollama import Ollama

# Use a local model to generate
Settings.llm = Ollama(
    model="llama3.1:8B",
    request_timeout=360.0,
    context_window=8000
    )

def main():
    # Load embedding model (same as used for vector store)
    embed_model = HuggingFaceEmbedding(model_name="all-mpnet-base-v2")
    Settings.embed_model = embed_model

    # Load persisted vector store + metadata
    storage_context = StorageContext.from_defaults(persist_dir="./storage")
    index = load_index_from_storage(storage_context)

    query_engine = index.as_query_engine()

    # Query
    while True:
        q = input("\nEnter your question (or 'exit'): ").strip()
        if q.lower() in ("exit", "quit"):
            break
        print()
        response = query_engine.query(q)
        print(response)

if __name__ == "__main__":
    main()

-UU-:---  F1  query.py       Top   L44    (Python ElDoc) ----------------------------------------------------------------------
```

RAG query in
~40 lines of code

See https://docs.llamaindex.ai/

Possible LLM pitfalls and problems

Hallucinations

LLMs sometimes generate plausible-sounding but factually incorrect or entirely fabricated information

Misalignment

The model's output may not reflect user intent or ethical/social expectations, especially in nuanced or sensitive contexts.

Bias and Stereotypes

LLMs can amplify societal biases, especially around race, gender, or culture, due to patterns in their training data.

Overconfidence

LLMs often present answers in a confident tone regardless of uncertainty, leading users to overtrust incorrect outputs.

Context Loss

When prompts are too long or complex, LLMs may forget or ignore earlier context, causing incoherent or inconsistent responses.

In our uses…

Hallucinated Code or APIs

Suggests functions, libraries, or syntax that don't exist

Misuse of Retrieved Content (RAG)

Pulls in irrelevant or misunderstood snippets, leading to wrong answers

Inconsistent Formatting

Breaks expected structure (e.g. JSON, YAML, code blocks) even in repeated tasks

Overconfident Errors

Gives incorrect results or logic with a confident tone

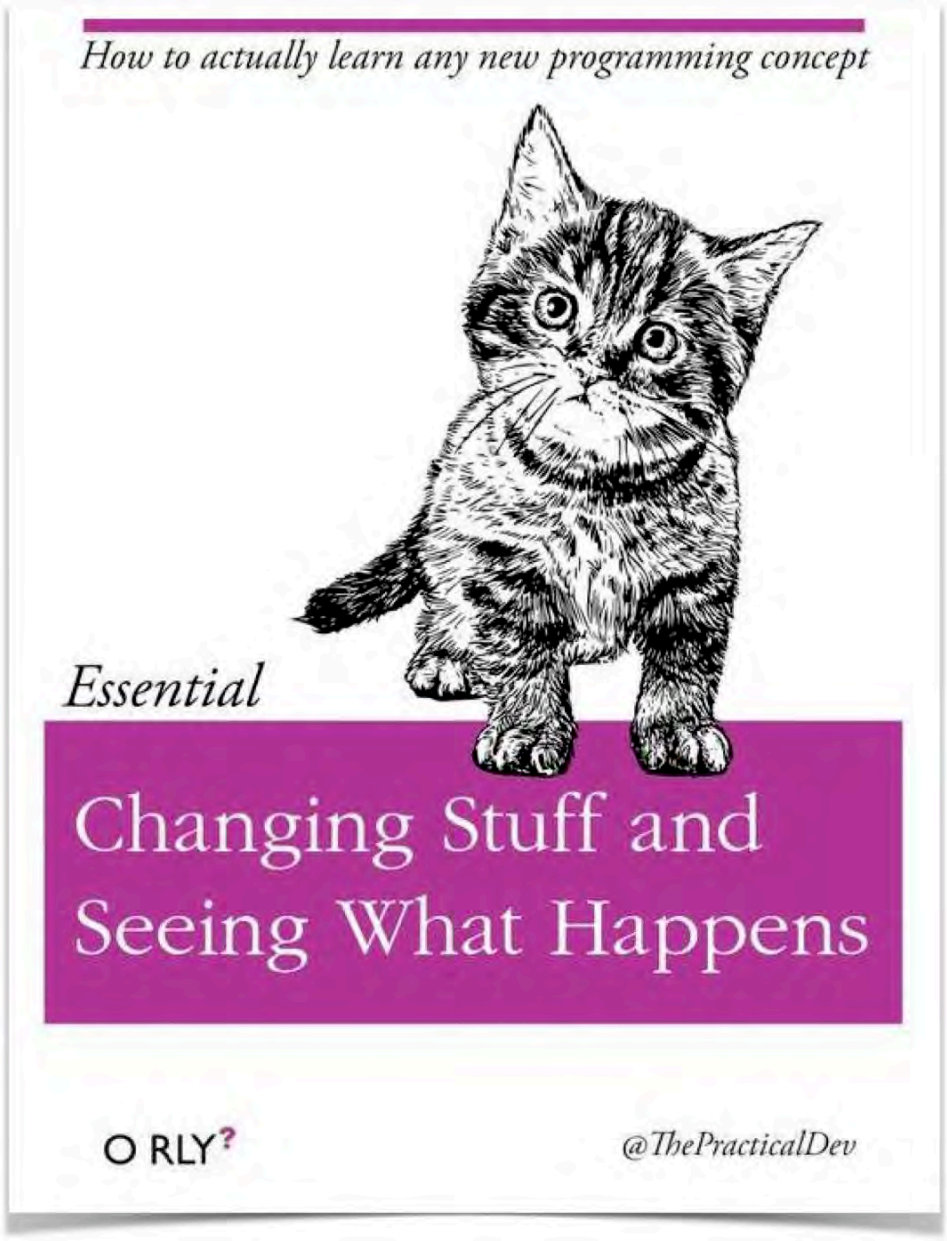
Context Limitations

Forgets earlier input in long prompts or complex workflows

"…all people using these systems should be informed and constantly reminded that their requests could lead to error-prone, fabricated, or otherwise misaligned responses* as well as potentially dangerous actions in the case of AI agents."

Why AI Chatbots Lie to Us
Melanie Mitchell, Science, July 26, 2025
https://doi.org/10.1126/science.aea3922

* may also apply to faculty



"Programming is a skill best acquired by practice and example rather than from books."
– Alan Turing, *Programmers' Handbook for Manchester Electronic Computer Mark II*, 1951

Learn by doing – hack on an LLM

HTTPS://WWW.YOUTUBE.COM/WATCH?V=KCC8FMEB1NY

Andrej Karpathy

@AndrejKarpathy · 605K subscribers · 16 videos

SuperThanks: very optional, goes to Eureka Labs. ...more

eurekalabs.ai and 4 more links

Subscribe

Home Videos Playlists Community

General Audience ▶ Play all

videos for more general audience, no programming experience necessary.

[1hr Talk] Intro to La
Language Models

Andrej Karpathy

2.4M views • 1 year ag

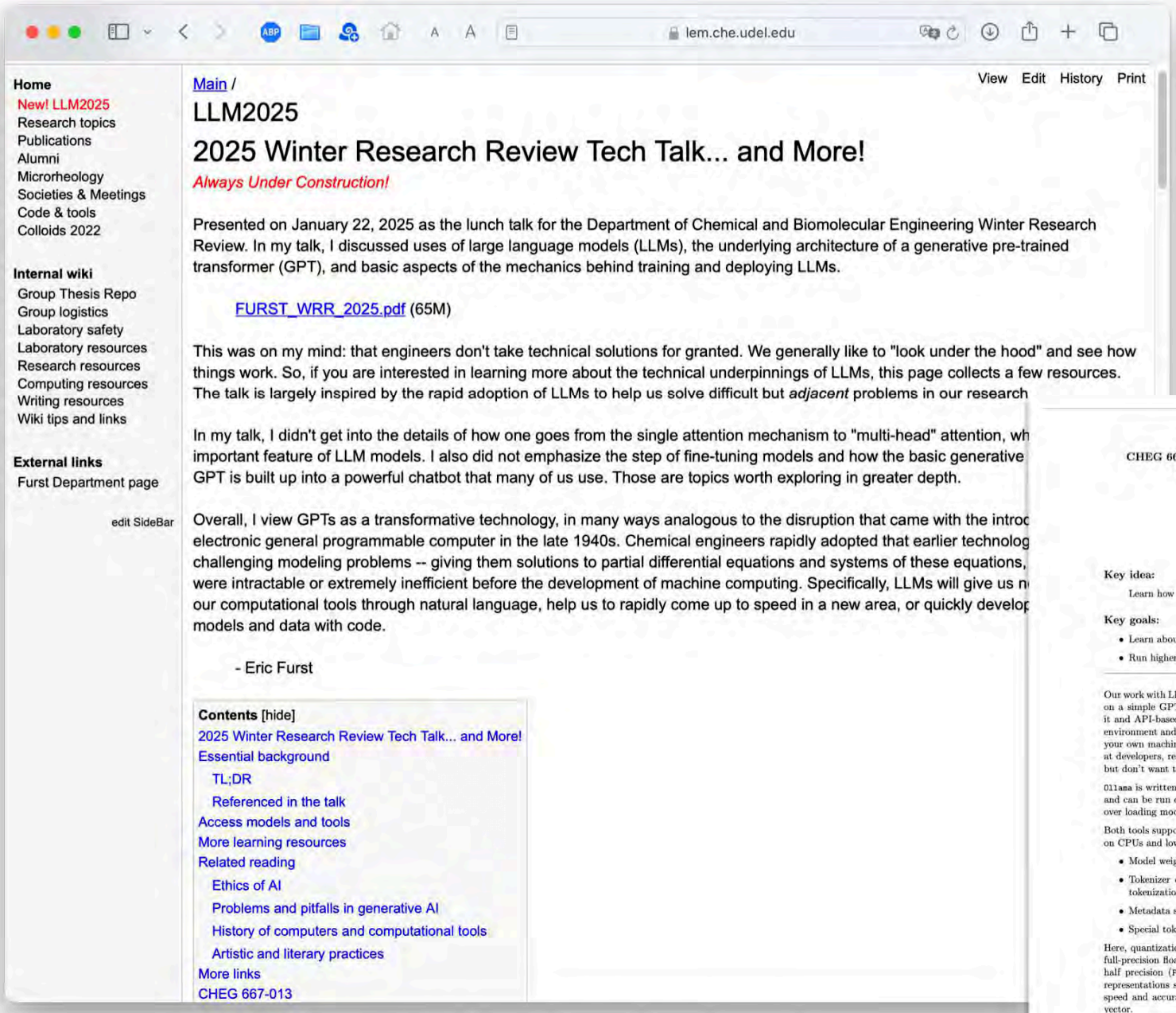
Neural Netwo

Let's build GPT: from scratch, in code, spelled out.

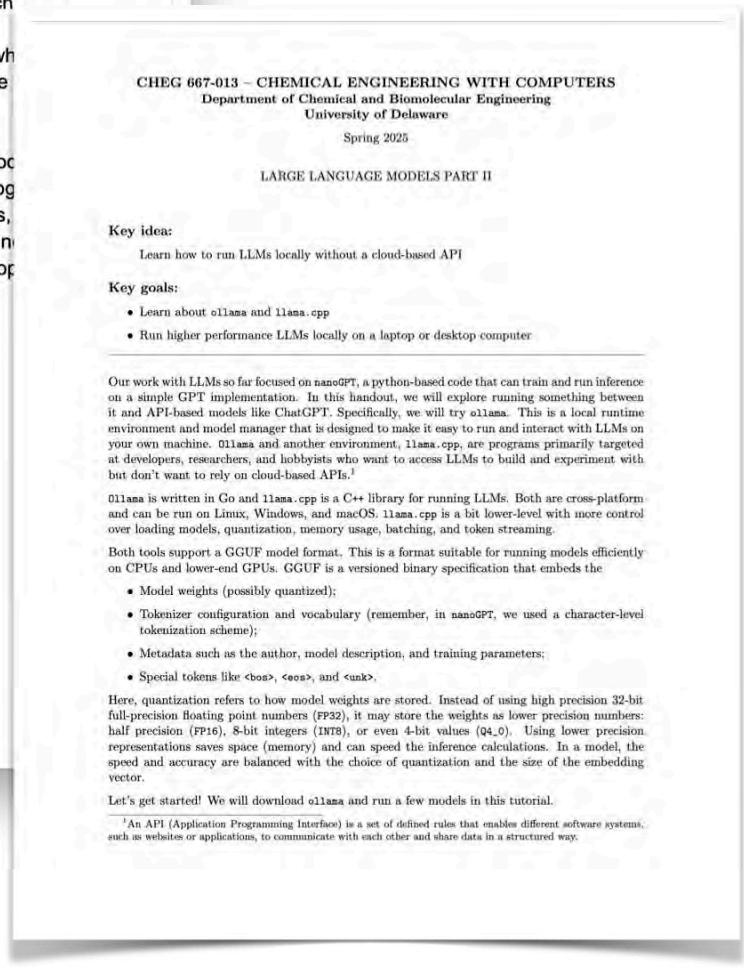
5M views • 2 years ago

```
furst@anisotropic:~/LLM/nanoGPT$ wc $(ls -1 *.py)
  117   487  4815 bench.py
   47   219  1758 configurator.py
  331  1798 16507 model.py
   93   522  4313 sample.py
  336  1799 14845 train.py
  924  4825 42238 total
```

~300 lines of Python
Runs on CPU or GPU



CHEG667-013 handouts!



CHEG 667-013 – CHEMICAL ENGINEERING WITH COMPUTERS
Department of Chemical and Biomolecular Engineering
University of Delaware

Spring 2025

LARGE LANGUAGE MODELS PART II

Key idea:

Learn how to run LLMs locally without a cloud-based API

Key goals:

- Learn about `ollama` and `llama.cpp`
- Run higher performance LLMs locally on a laptop or desktop computer

Our work with LLMs so far focused on `nanoGPT`, a python-based code that can train and run inference on a simple GPT implementation. In this handout, we will explore running something between it and API-based models like ChatGPT. Specifically, we will try `ollama`. This is a local runtime environment and model manager that is designed to make it easy to run and interact with LLMs on your own machine. `Ollama` and another environment, `llama.cpp`, are programs primarily targeted at developers, researchers, and hobbyists who want to access LLMs to build and experiment with but don't want to rely on cloud-based APIs.[1]

`Ollama` is written in Go and `llama.cpp` is a C++ library for running LLMs. Both are cross-platform and can be run on Linux, Windows, and macOS. `llama.cpp` is a bit lower-level with more control over loading models, quantization, memory usage, batching, and token streaming.

Both tools support a GGUF model format. This is a format suitable for running models efficiently on CPUs and lower-end GPUs. GGUF is a versioned binary specification that embeds the

- Model weights (possibly quantized);
- Tokenizer configuration and vocabulary (remember, in `nanoGPT`, we used a character-level tokenization scheme);
- Metadata such as the author, model description, and training parameters;
- Special tokens like `<bos>`, `<eos>`, and `<unk>`.

Here, quantization refers to how model weights are stored. Instead of using high precision 32-bit full-precision floating point numbers (`FP32`), it may store the weights as lower precision numbers: half precision (`FP16`), 8-bit integers (`INT8`), or even 4-bit values (`Q4_0`). Using lower precision representations saves space (memory) and can speed the inference calculations. In a model, the speed and accuracy are balanced with the choice of quantization and the size of the embedding vector.

Let's get started! We will download `ollama` and run a few models in this tutorial.

[1] An API (Application Programming Interface) is a set of defined rules that enables different software systems, such as websites or applications, to communicate with each other and share data in a structured way.

https://furst.group

Concluding remarks

1. How LLMs work
2. Running LLMs locally
3. Two (or three) LLM uses

Solving *adjacent* problems in our science:
coding, data cleaning, formatting

New ways to build computational tools with computer-human interactions through natural language

RAG (search, information retrieval) that uses the encoding / embedding power of transformers