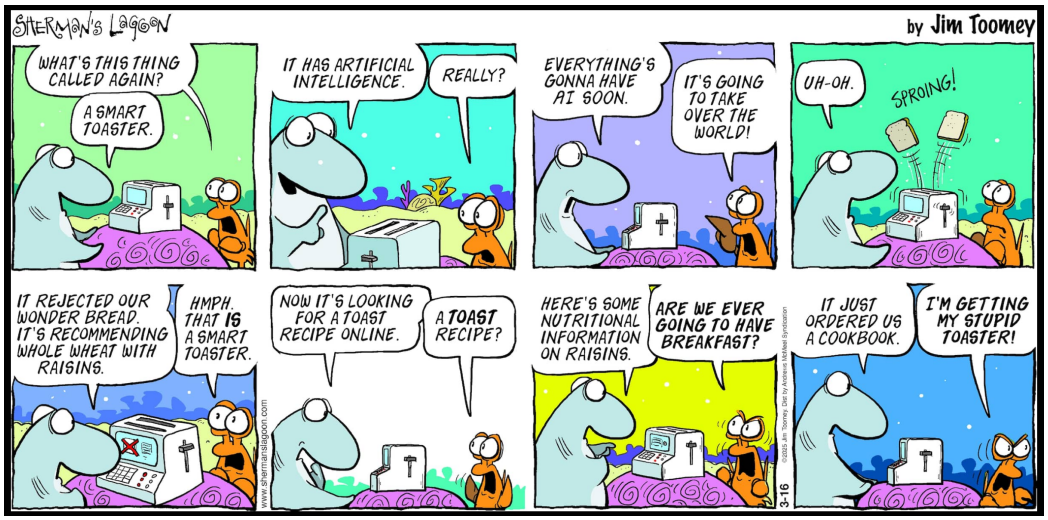
CSCI 3202

Lecture 37

November 21, 2025



Sherman's Lagoon by Jim Toomey. http://www.shermanslagoon.com/

Announcements

- No Quiz today!
- FCQs are available now
  - Please take time to review this class
  - Your reviews have a big impact on my continued employment
- Homework 8 is due on Wednesday, 12/3 by 11:59 pm
  - Fitting a decision tree model
  - There are code fragments that help you with the work
  - Final homework for semester
  - Remember that you can drop **one** homework
- Interview grading available now - Link for signup available on Canvas
  - 15 minute interviews
  - Interviews are done on Zoom
  - Share your IDE
  - Run your project and answer questions
  - We will do interview grading after fall break as well
- You may resubmit Worked Out Problems (Q13-Q20) only from the midterm for regrading. The points you receive will replace your original score on the

midterm. If you do not submit anything, there will be no change in your score. You have until Friday November 21 to submit. No late submissions will be accepted

- Before calculating your final grade, I will add 5 points to your midterm exam score due to the length of the exam
- These points will only count towards your midterm score. They will not count as extra credit toward your course grade

Final Exam

- Will be remote ("Take Home") on Canvas
- Time limited to 3 hours once you open the exam
- On paper and in Canvas
- Similar in format to Midterm
- Take a photo of your results and turn in photo(s)
- Comprehensive, but focused mainly on last half of semester
- Will be due at the listed time of our Final, Thursday at 4:00 pm
- No late exams will be accepted

Summary on how ChatGPT works

- **Data:** It trains on huge amounts of text.
- **Pattern Recognition:** It learns statistical patterns in language.
- **Algorithm:** Uses a Transformer to predict the next word/token.
- **Abstraction:** It represents meaning using thousands of learned internal features.
- **Feedback Loops:** Humans guide it to be more helpful and safe.
- **Limitation Awareness:** It doesn't "understand" like humans—it generates text probabilistically.

Deeper Dive on ChatGPT architecture

- https://www.zdnet.com/article/how-chatgpt-actually-works-and-why-its-been-so-game-changing/

What Is a Transformer?

A **Transformer** is a neural network architecture designed to understand and generate sequences—like text—by paying attention to relationships between words. It replaced older models like RNNs and LSTMs, enabling much better long-range reasoning and scalability.

Transformers are the foundation of GPT (“**G**enerative **P**retrained **T**ransformer”).

Key Ideas Behind Transformers

1. Tokens

Text is broken into pieces called *tokens* (words, subwords, or characters).

Example:

“ChatGPT is powerful” → `["Chat", "G", "PT", " is", " powerful"]`

All Transformer processing happens on these token embeddings.

---

2. Self-Attention: Finding What Matters

The heart of a Transformer is **self-attention**.

For every token, the model asks:

> ***“Which other tokens in this sentence are important for understanding me?”***

Each token creates:

- a **Query** vector
- a **Key** vector
- a **Value** vector

Attention works like this:

1. Compare Query of token A with Keys of all other tokens.
2. Compute relevance scores (attention weights).
3. Mix the Value vectors according to those weights.

This lets the model learn patterns like:

- pronoun references
- grammar relationships
- long-distance dependencies
- context and meaning

Unlike older models, attention can look at *any position in the sequence* instantly.

---

3. Multi-Head Attention

Instead of one attention pattern, Transformers use *many heads* in parallel.

Each head specializes in something different, e.g.:

- syntactic structure
- long-range relations
- coreference
- topic tracking

Their outputs are combined for a richer understanding.

---

4. Feed-Forward Networks

After self-attention, each token passes through a small neural network that transforms its representation.

These layers help the model learn nonlinear and abstract patterns.

---

5. Stacking Layers

Transformers consist of many repeated layers (ChatGPT uses dozens or hundreds).

Each layer refines the representation of each token.

Early layers: syntax and local patterns

Middle layers: meaning and structure

Late layers: high-level reasoning, world knowledge

---

6. Positional Encoding

Attention has no sense of order, so Transformers add **positional encodings** that tell the model where tokens appear in the sequence.

---

7. Training: Predict the Next Token

The base GPT model is trained using a simple task:

> ***Predict the next token given all previous tokens.***

Example:

Input: “The cat sat on the”

Target: “mat”

By doing this trillions of times across huge datasets, the model learns:

- grammar
- facts
- reasoning patterns

- writing styles
- logic and relationships

---

8. Generation: Sampling the Next Token

When responding to you, ChatGPT:

1. Looks at everything you've said.
2. Computes attention across the whole context.
3. Predicts the most likely next token.
4. Adds it to the sequence and repeats.

This produces fluent, context-aware responses token-by-token.

Putting It All Together

A Transformer lets ChatGPT:

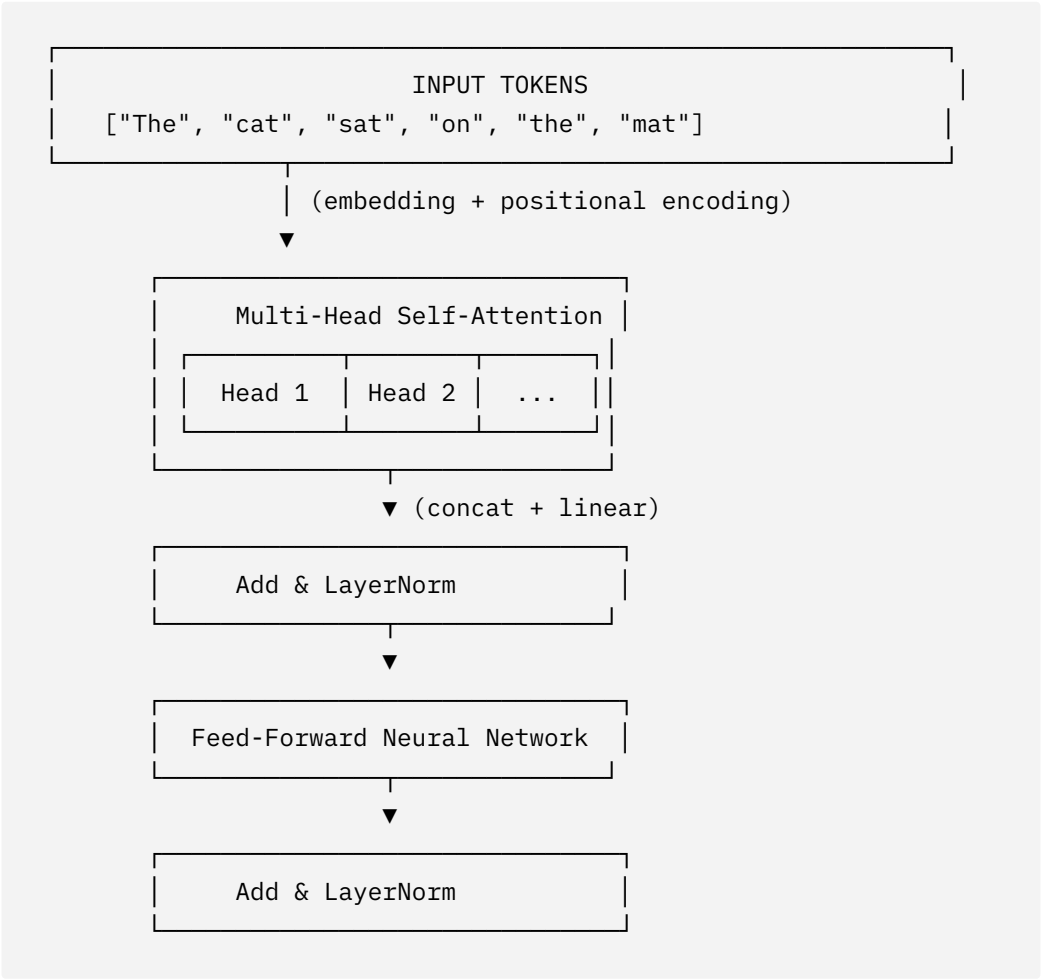
- read your entire message at once
- understand relationships between all words
- use learned knowledge to predict a good response
- generate text one token at a time

It's fast, scales extremely well, and supports large amounts of context—making modern AI chat systems possible.

An Example

1) 📊 A Simple Diagram of a Transformer (Encoder-style block used by GPT)

(GPT uses only decoder blocks, but the core attention mechanism is the same.



GPT stacks **dozens to hundreds** of these blocks.

---

2) 🧪 A Step-by-Step Example With Real Tokens

Let's walk through a *tiny*, simplified example.

We'll use this sentence:

> ***"The cat sat on the mat"***

Tokenized (example using simplified tokens):

```
["The", "cat", "sat", "on", "the", "mat"]
```

👉 Step 1 — Convert Tokens to Embeddings

Each token becomes a numerical vector (pretend each is 4-dimensional for simplicity):

```
The → [0.1, 0.4, 0.3, 0.6]
cat → [0.9, 0.2, 0.1, 0.5]
sat → [0.7, 0.1, 0.8, 0.2]
on  → [0.2, 0.7, 0.3, 0.4]
the → [0.1, 0.4, 0.3, 0.6]   (same token → same embedding)
mat → [0.8, 0.1, 0.9, 0.1]
```

Add positional encodings to provide order info.

👉 Step 2 — Build Q, K, V Vectors

Each embedding is multiplied by learned weight matrices:

```
Query = Embedding × W_Q
Key   = Embedding × W_K
Value = Embedding × W_V
```

(These are learned during training.)

So for “cat”, for example:

```
Embedding(cat) × W_Q → Q_cat
Embedding(cat) × W_K → K_cat
Embedding(cat) × W_V → V_cat
```

You'll get new vectors such as:

```
Q_cat = [1.2, -0.3, 0.5]
K_the = [0.4,  0.2, 1.1]
V_sat = [0.7, -0.1, 0.3]
```

👉 Step 3 — Compute Attention Scores

To compute how much **“cat”** pays attention to each other token:

Score = dot(Q_cat, K_x)

Example (totally simplified numbers):

```
Score(cat → The) = 1.1
Score(cat → cat) = 2.4
Score(cat → sat) = 0.3
Score(cat → on ) = 0.8
Score(cat → the) = 1.2
Score(cat → mat) = 0.5
```

👉 Step 4 — Softmax the Scores

This turns raw scores into probabilities:

```
softmax([1.1, 2.4, 0.3, 0.8, 1.2, 0.5])
   = [0.18, 0.50, 0.03, 0.09, 0.16, 0.04]
```

Interpretation:

- “cat” attends **50%** to itself
- **18%** to “The”
- **16%** to the second “the”
- small attention to other tokens

(Actual models have many heads, each learning different patterns.)

👉 Step 5 — Weighted Sum of Values

Now mix the **Value** vectors according to those attention weights:

```
Output(cat) =
   0.18 * V_The +
   0.50 * V_cat +
   0.03 * V_sat +
   0.09 * V_on  +
   0.16 * V_the +
   0.04 * V_mat
```

This produces a new representation of the word **“cat”** that incorporates contextual meaning.

Every token does this simultaneously.

👉 Step 6 — Multi-Head Attention

Several attention processes run in parallel:

- Head 1 might track grammar
- Head 2 might track semantic roles
- Head 3 might track longer context
- Head 4 might track sentence structure

Their outputs are concatenated.

👉 Step 7 — Feed-Forward Network

Each token’s resulting vector goes through a small neural network:

```
output = GELU(W2 · GELU(W1 · x))
```

This gives the model nonlinear expressive power.

👉 Step 8 — Repeated Layers

GPT uses **many** layers like this—each one refining its internal understanding.

Early layers: word relationships

Mid layers: meaning, subject–object relationships

Late layers: reasoning, planning, world knowledge

👉 Step 9 — Predict the Next Token

After processing the whole context, the model produces a probability distribution for the next token.

For the sequence:

```
"The cat sat on the"
```

The model might output probabilities like:

```
mat: 0.87
```

```
floor: 0.03
ground: 0.02
sofa: 0.01
...
```

GPT selects (or samples) a token — **“mat”** — and repeats.

More Details

- Transformer architecture
  - https://www.datacamp.com/tutorial/how-transformers-work
  - https://www.geeksforgeeks.org/deep-learning/architecture-and-working-of-transformers-in-deep-learning/
  - Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., ... & Polosukhin, I. (2017). Attention is all you need. Advances in neural information processing systems, 30.

Next Week

- Finish up on ChatGPT
- Begin preparing for Final Exam
- I hope you have an enjoyable Fall break!