Large Language Models (LLMs): Foundations, Architectures, and Applications

Module 1 – Introduction and Foundations

Large Language Models (LLMs)

- What are LLMs and why do they matter?
- Course roadmap: architecture, training, applications, ethics
- Learning goals for this module:
  - Understand why LLMs emerged
  - Learn how embeddings and transformers work
  - Recognize key architectural components

From Rules to Representation

- 1950s–1970s: Symbolic AI → handcrafted rules
- 1980s–1990s: Statistical NLP → n-grams, HMMs
- 2000s: Word embeddings (Word2Vec, GloVe)
- 2017: Transformer architecture revolution ("Attention is All You Need")
- 2020s: GPT, PaLM, LLaMA, Mistral – scaling laws and emergent abilities

Why LLMs?

- Capture context and meaning beyond local windows
- Learn representations directly from massive text corpora
- Replace manual feature engineering with end-to-end learning
- Enable transfer learning → pre-train once, fine-tune everywhere

The Shift in Representation Learning

- Word Embeddings: static vectors → semantic similarity
- “king – man + woman ≈ queen”
- Contextual Embeddings: depend on sentence context (ELMo, BERT)
- Transformers: learn contextual embeddings through attention

Self-Attention: Context through Comparison

- Queries, Keys, and Values
- Weighted averaging of words → context-aware meaning
- Multi-head attention → multiple relational “views”

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

Self-Attention: Context through Comparison

- When a model reads a sentence like

“The cat sat on the mat”

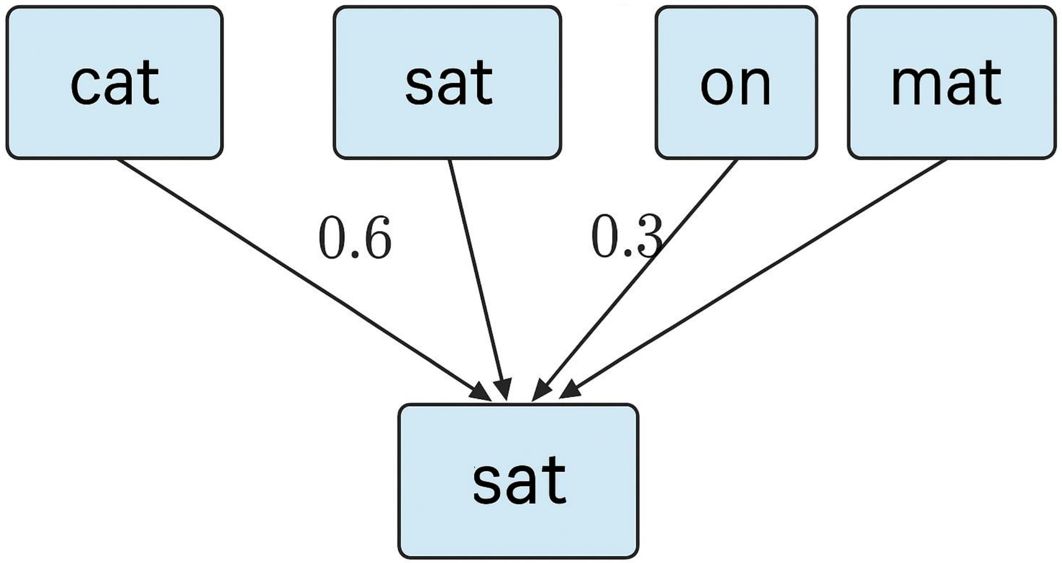
- it needs to understand which words are related to each other.
- For example, the word “sat” cares about “cat” (who sat) and “mat” (where it sat).
- That’s what attention does — it helps the model decide which words to focus on.

Self-Attention: Context through Comparison

- Let’s see how “sat” attends to other words.
- Query = “sat”
- Keys = [“The”, “cat”, “sat”, “on”, “the”, “mat”]
- Compute similarity: “sat” is most related to “cat” and “mat”.
- softmax gives higher weights to “cat” and “mat”.
- Multiply those weights by each word’s value (V) → result = context vector for “sat”.

Self-Attention: Context through Comparison

- The model now knows that the meaning of “sat” is mostly influenced by “cat” and “mat” — not by “the” or “on”.
- That’s how attention lets the model understand context — who’s doing what, to whom, and where — without needing to read the sentence one word at a time like older models (RNNs).



Terminology

Term	Definition (Simplified)	Analogy / Mental Model	Example
Embedding	Numerical vector representing a word or token’s meaning.	Like a GPS coordinate for a word — similar meanings are close together.	embedding("king") ≈ embedding("queen")
Transformer	Neural network using attention instead of recurrence.	Like a team of readers who all read the book and share notes.	GPT, BERT, Mistral models.
Token	Smallest text unit (word, subword, or character).	Like Lego blocks building sentences.	“playing” → ["play", "##ing"]
Parameter	A numerical weight learned by the model.	Like knobs on a sound mixer tuning the final output.	GPT-3 has **175B parameters**.

Terminology

Term	Definition (Simplified)	Analogy / Mental Model	Example
Self-Attention	Each word looks at others to find what's relevant.	Like students in a group discussion listening to each other.	"sat" attends to "cat" and "mat."
Multi-Head Attention	Several attention layers work in parallel, each focusing on different relations.	Like many spotlights highlighting different parts of a play.	One head tracks subject–verb; another, adjective–noun.
Query, Key, Value (QKV)	Used to measure how relevant each token is to others.	Like a search engine: Query = question, Key = title, Value = content.	"I" (query) looks at "love NLP" (keys/values).
Attention Weights	Scores showing how much focus one token gives to another.	Like eye contact in conversation — who you're paying attention to.	Visualized as attention heatmaps.

Terminology

Term	Definition (Simplified)	Analogy / Mental Model	Example
Positional Encoding	Adds order info to embeddings since Transformers read in parallel.	Like page numbers or timestamps showing sequence.	“dog bites man” ≠ “man bites dog.”
Residual Connection	Shortcut that adds input to output, keeping info stable.	Like a safety rope for a climber.	output = x + layer(x)
Feed-Forward Network (FFN)	Two-layer network refining attention results per token.	Like a chef tasting and adjusting each dish.	Linear → ReLU → Linear.
Normalization (LayerNorm)	Keeps layer activations stable during training.	Like a thermostat keeping temperature steady.	LayerNorm(x) prevents exploding gradients.

Terminology

Term	Definition (Simplified)	Analogy / Mental Model	Example
Encoder	Reads input text and builds contextual embeddings.	Like a summarizer understanding a paragraph.	Used in BERT and ViT.
Decoder	Generates output text token by token.	Like a storyteller retelling what was understood.	Used in GPT and T5.
Contextual Embedding	Word meaning changes based on sentence context.	Like a chameleon changing color with its surroundings.	“bank” near “river” vs. “bank” near “money.”
Attention Output (Context Vector)	Weighted sum of all words based on attention scores.	Like a blended smoothie of all relevant info.	“sat” receives most info from “cat” and “mat.”

Terminology

Word	Embedding Vector (simplified)	Closest Neighbors
"dog"	[0.61, -0.28, 0.45]	cat, puppy, pet
"car"	[0.10, 0.82, -0.13]	truck, vehicle, bus
"apple"	[-0.72, 0.40, 0.33]	orange, fruit, banana

Embedding

```python
sentences = ["I love NLP", "I love AI", "NLP loves AI"]

from sklearn.feature_extraction.text import CountVectorizer
vec = CountVectorizer()
X = vec.fit_transform(sentences)        # rows = sentences, cols = words

print(vec.get_feature_names_out())      # vocabulary order
print(X.toarray())                      # sentence embeddings
```

```
['ai' 'love' 'loves' 'nlp']
[[0 1 0 1]
 [1 1 0 0]
 [1 0 1 1]]
```

Embedding

```
corpus = [
  "the cat sat on the mat",
  "the dog sat on the rug",
  "the cat chased the mouse"
]
tokenized = [s.split() for s in corpus]

from gensim.models import Word2Vec
model = Word2Vec(
    sentences=tokenized,
    vector_size=50,    # embedding dimension
    window=5,          # context window
    min_count=1,       # keep all words in this tiny demo
    sg=1,              # 1=skip-gram (good for small data), 0=CBOW
    epochs=200
)

vec_cat = model.wv["cat"]                 # 50-dim vector
print(vec_cat[:5])                        # peek at first 5 numbers
print(vec_cat)

print(model.wv.similarity("cat", "dog"))    # closer ≈ more similar
print(model.wv.most_similar("cat", topn=3)) # nearest neighbors
```

```
[-0.01743123  0.00739823  0.01028193  0.01172741  0.0148505 ]
[-0.01743123  0.00739823  0.01028193  0.01172741  0.0148505  -0.01254173
  0.00266343  0.01249782 -0.00596588 -0.01219991 -0.00072718 -0.01670636
 -0.01141626  0.01420146  0.00664568  0.01446455  0.01390477  0.01532046
 -0.00792779 -0.00137248  0.00504984 -0.00895459  0.0171903  -0.01967666
  0.01378441  0.0056403  -0.0097696   0.00912248 -0.00386569  0.01313612
  0.0196351  -0.00869995 -0.00078545 -0.01167672  0.00757158  0.00548977
  0.01429759  0.01181683  0.01905415  0.01850919  0.01549578 -0.01386036
 -0.01854655 -0.00090891 -0.00582287  0.01597248  0.01166192 -0.00305323
  0.00308315  0.00394295]
0.092894495
[('mouse', 0.16969653964042664), ('the', 0.14504680037498474), ('chased', 0.14229169487953186)]
```

Open Questions

- Embedding Size
- Number Size
- Tokenizers