# Normalization in LLMs

- Normalization in LLMs is typically applied between the attention and Feed forward layers (and vise versa) its usally applied before or after each sub layer (depending on verients):
  - post norm: normalization after Attention/MLP (used in original transformer paper)
  - pre norm: normalization before Attention/MLP (used in modern LLMs as it stabolizer training in deep models)

Why? :
- prevents exploding/vanishing activations: means the numbers (output of layers) dont become too big or small as they pass through many layers - this keeps the models computation stable
- Makes optimizations smoother: means the model learns more stedily during training, the loss surface becomes less chaotic, so gradient decent can take more reliable steps towords better preformance insted of bouncing around/getting stuck
- Allows deeper, more stable transformer training: beacuse norm keeps each layers well behaved (balanced and stable not too big or small, consistant across layers) you can stack more layers without the models training becoming unstable or diverging, in short normalization makes it possible for very deep NN like LLMs to still learn effectively and converge smoothly, insted of breaking down as they grow in size

(for rag see 94,93,106)
(for vector DB see 95)

## Vector Databases vs knowlage graph and usage in RAG

- Knowlage graphs: are a structured representation of facts and relationships, it stores entities (nodes) and relastionships (edges) → like a map of how concepts connect, each entity has attributes and connections, allowing resoning, querying and context linking
  Ex) (Elon) –[founded]→ (Tesla)
  (Tesla) –[Makes]→ (EV's)
  ⋯→ node ⋯→ edge
- Vector DB: store embeddings (numaric repersentations of text, imgs, etc) it allows sematic search by simmilarity (cos, Euclidian). Ex:
  - you embed a document into a 1536-dim vector
  - you query "who made tesla". DB finds most simantically simmillar chunks (even if words differ)

| Feature | Knowlage graph | Vector Databases |
|---|---|---|
| structure | Explicit (nodes, edges, relations) | implicit (numeric emb) |
| Query type | Symbolic/logical (graph traversal) | Semantic/simmilarity serch |
| Strength | Precise resoning and explonibility | Flexible meaning based ret |
| Data type | structured facts | Unstructerd media, text |
| Ex | Neo4J, AWS Neptune | Pinecone, Weavite, FAISS |

- IN RAG: you can use either or Both, in classic RAG (Vector DB) you retrive semantically simmillar chunks from embeddings and feed it to LLM, in KG-RAG you retrieve related entites/facts via graph traversal and or Augmented Emb (embeddings enhanced with additional context or metadata) with graph links
  Ex: use KG to find connected info ("founder of EV company) then vector DB to fetch text about them or use Hybird RAG; use Vetor DB - semantic Recall, KG for logic, resoning, structure

Ex) KG = structerd brain of facts, used in searchengines
VectorDB = memory of meanings, used in RAG pipelines

- They complement each other, not replacements