nasscom
presents

Developer Workshop

Code, Create, Collaborate: Unlock Your Potential!

June 25, 2025

09:00AM- 04:00PM

Persistent Ramanujan, Hinjawadi, Pune

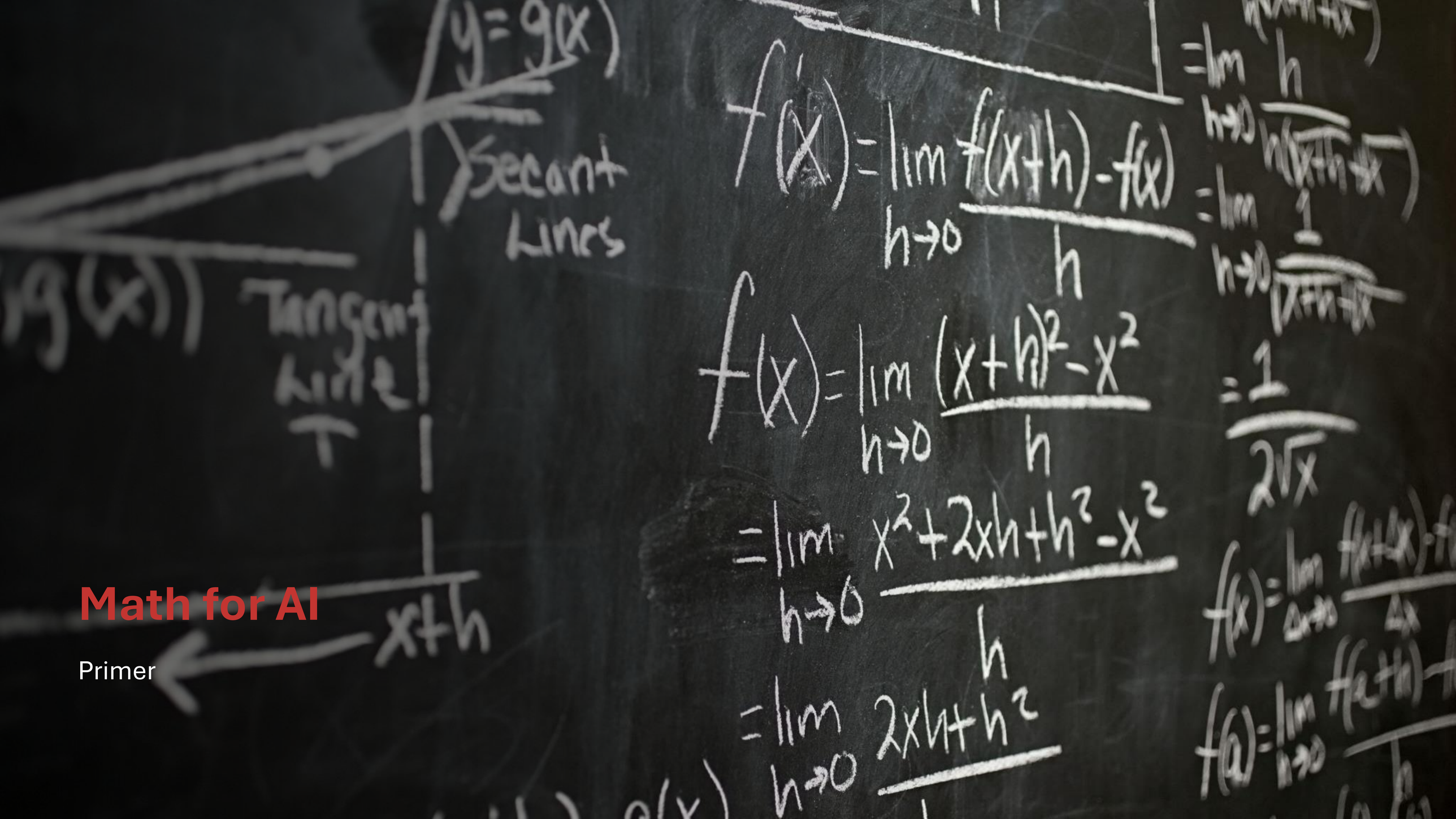
In Collaboration with

Persistent

nasscom

Smart Solutions: From Concept to Code

Chockalingam – Director - DS | AI | Technology

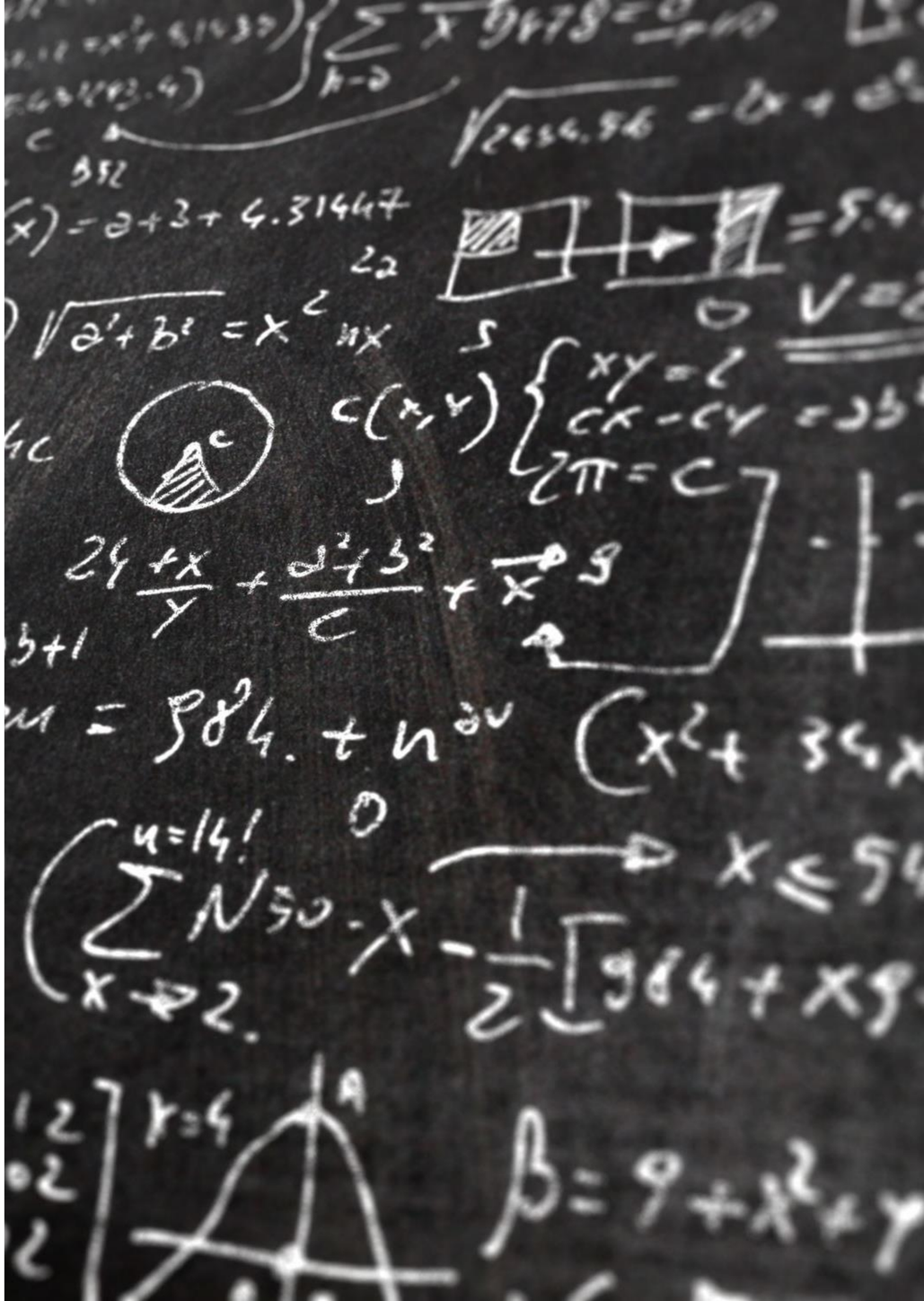


Math for AI

Primer

What you should know

- Linear Algebra – Vectors, Matrices, Tensors, Eigen Vectors and Eigen Values, Singular Value Decomposition, Principal Component Analysis.
- Calculus – Derivatives, vector/Matrix Calculus, Gradient Algorithms
- Statistics and Probability – Basic Stats, Basic Rules in Probability, Random Variables, Bayes Theorem, Maximum Likelihood estimation, Common Distributions
- Information Theory Concepts – Entropy, Cross Entropy, Kullback Leibler Divergence, Viterbi Algorithm, Encoder Decoder Algorithm



Reference Books and Videos

- Books

Linear Algebra – Gilbert Strang

Calculus - Calculus for Machine Learning Understanding the Language of Mathematics

Statistics and Probability – Anirban Das Gupta

Information Theory Concepts – David J C Mackey

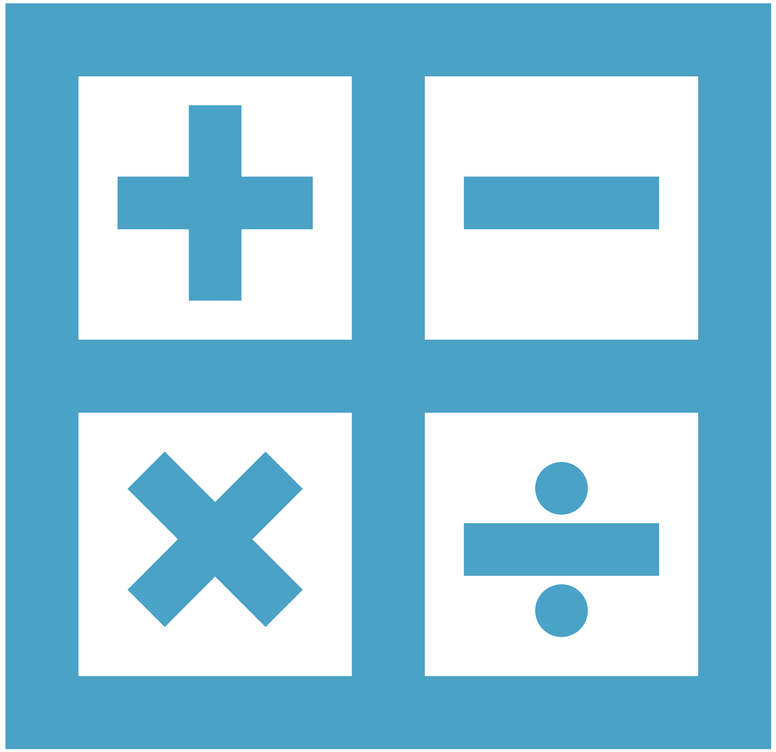
- Video Links
- https://www.youtube.com/watch?v=KzH1ovd4Ots
- https://www.youtube.com/watch?v=I-tmjGFaaBg

PyTorch Framework

Leading Deep Learning Framework

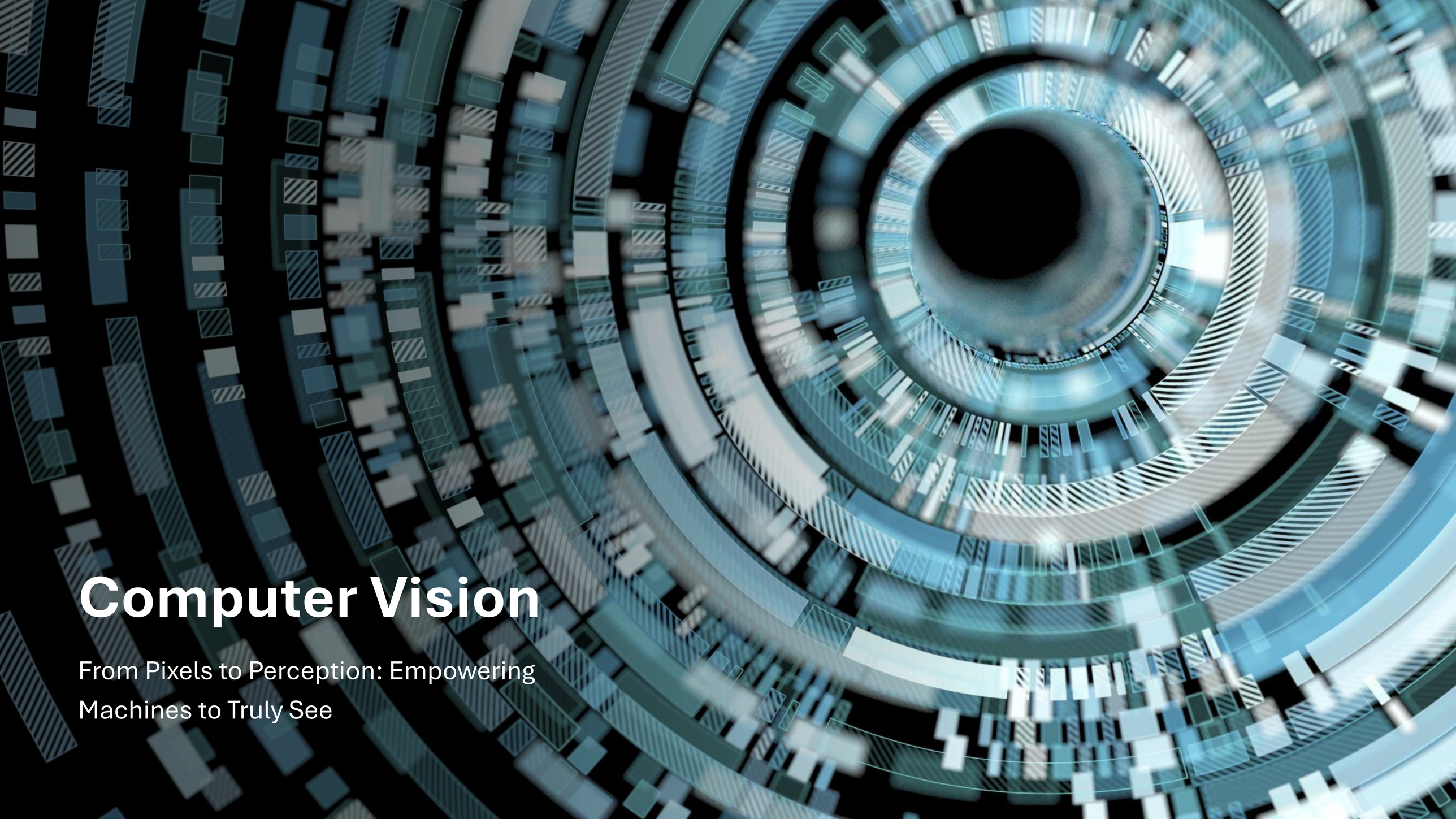
What to know

- PyTorch Fundamentals – Tensor data types, finding min Max and sum of Tensors, Reshaping, viewing and Stacking, squeezing and unsqueezing, permuting Tensors, Different ways of accessing GPUs in PyTorch.
- PyTorch Workflow – Linear Regression, predictions using inference mode, Training a Model. Setting up the optimizer and loss functions, Training Loops
- PyTorch Neural Network Classification
- PyTorch Computer Vision
- PyTorch Transfer Learning
- PyTorch Model Deployment



Popular Pytorch Courses

- https://www.youtube.com/watch?v=GIsg-ZUy0MY
- https://pytorch.org/tutorials/beginner/introyt/introyt_index.html

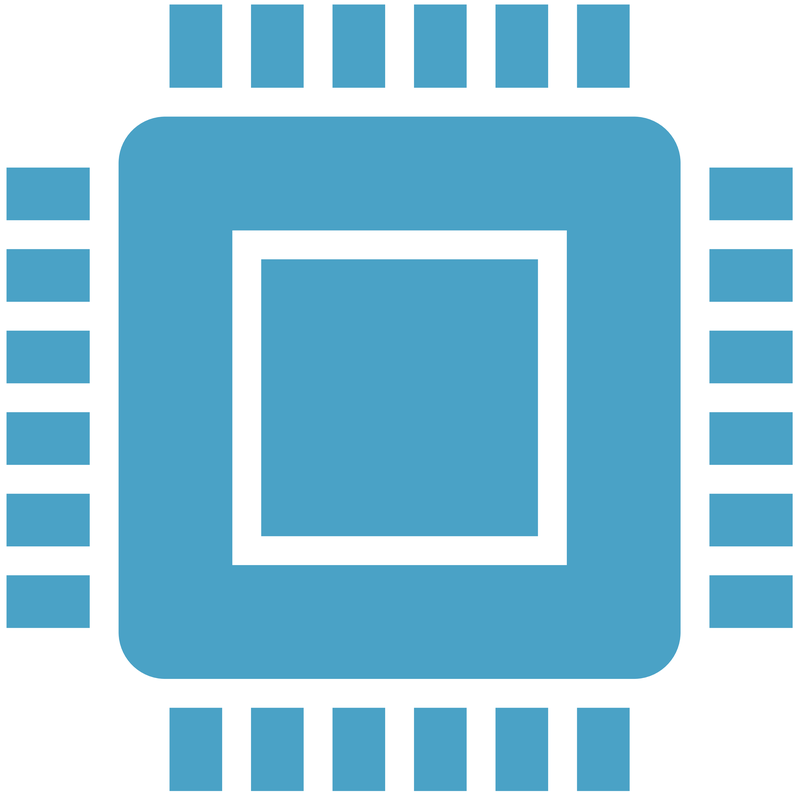


Computer Vision

From Pixels to Perception: Empowering Machines to Truly See

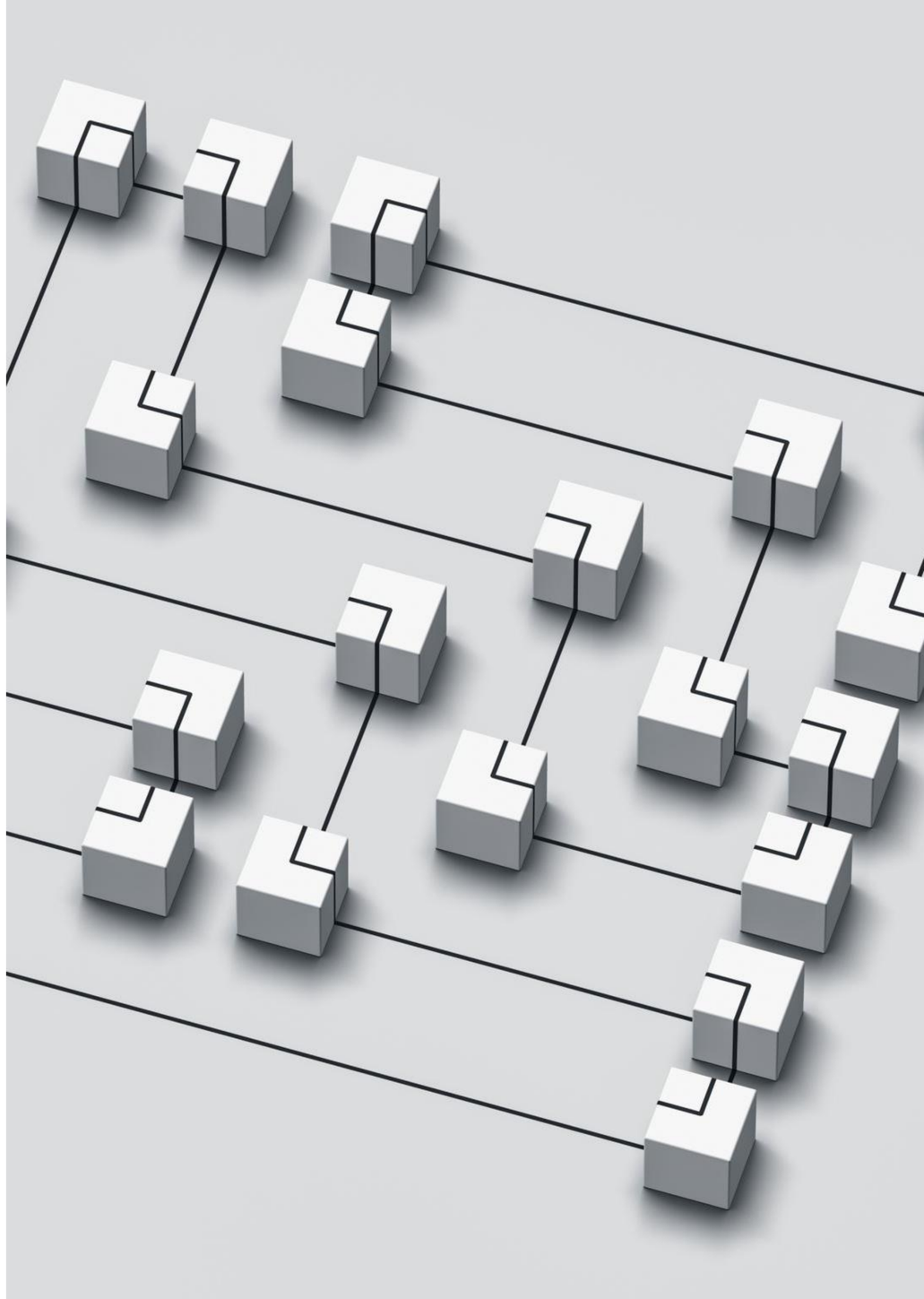
What is it?

- Enabling computers to interpret and understand digital images and videos.
- Automate tasks that the human visual system does—recognition, detection, tracking, etc.
- Core Techniques
  - Image Classification: Identifying what is in the image.
  - Object Detection: Locating and classifying multiple objects within an image.
  - Segmentation: Partitioning an image into regions or objects (e.g., semantic or instance segmentation).
  - Feature Extraction & Matching: Using descriptors (e.g., SIFT, ORB) for tasks like image stitching or 3D reconstruction.



Process

- Data Collection: Gathering diverse and representative datasets.
- Preprocessing: Normalizing images, augmenting data, handling noise.
- Modelling: Using traditional methods (e.g., classical filters, feature engineering) or deep learning (Convolutional Neural Networks, Transformers).
- Training & Validation: Iterative optimization of model weights on labelled datasets.
- Inference & Deployment: Applying trained models on new, unseen data in real-world applications.



Applications

Autonomous Vehicles (lane and object detection)

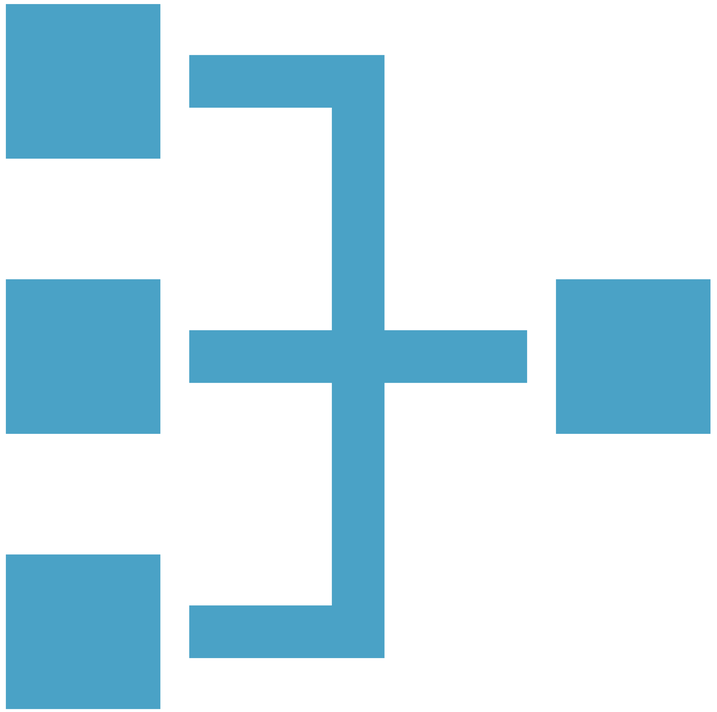
Medical Imaging (tumor detection, pathology & Radiology)

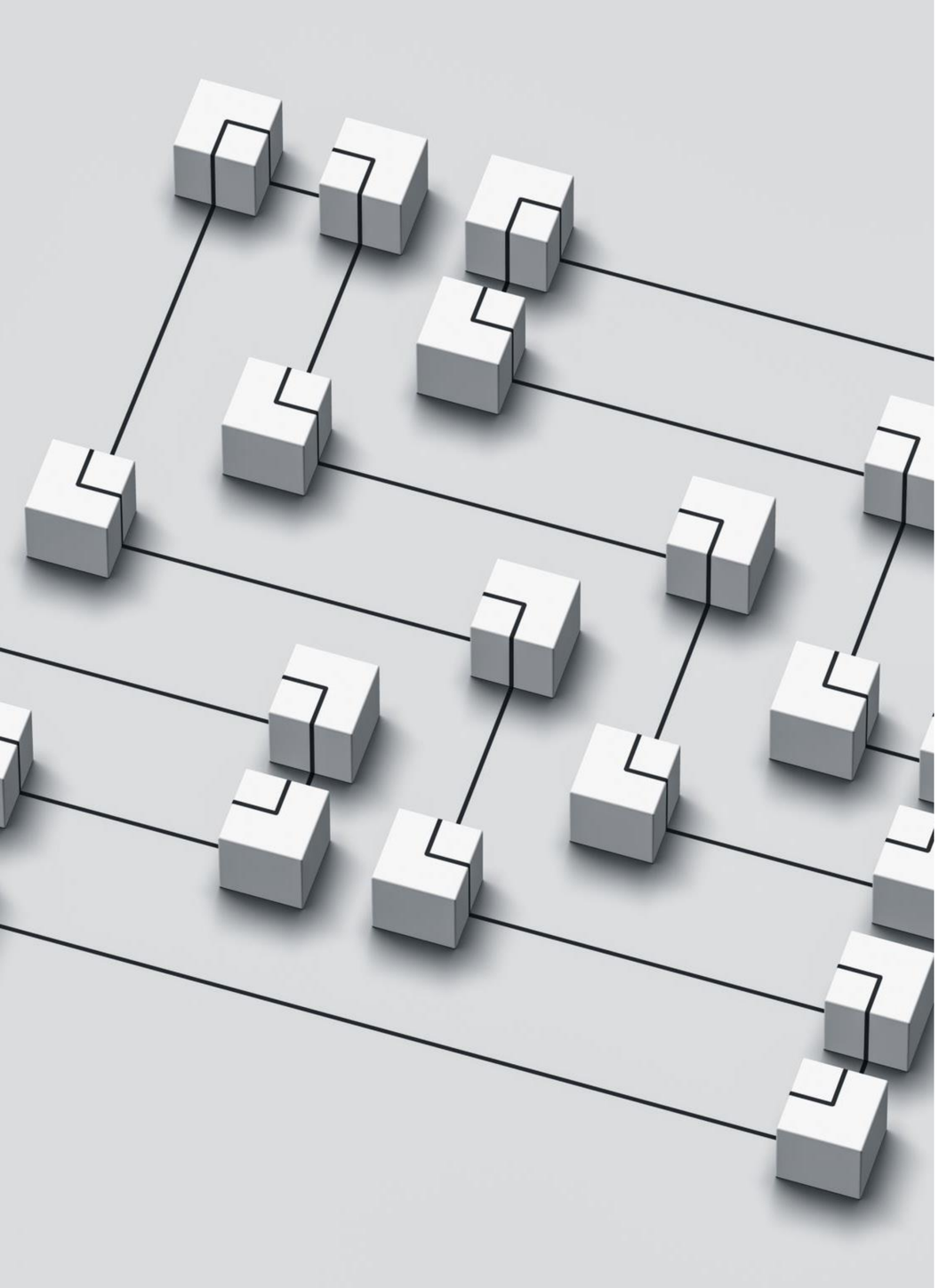
Surveillance & Security (face recognition)

Industrial Automation (quality control, robotics)

YOLO

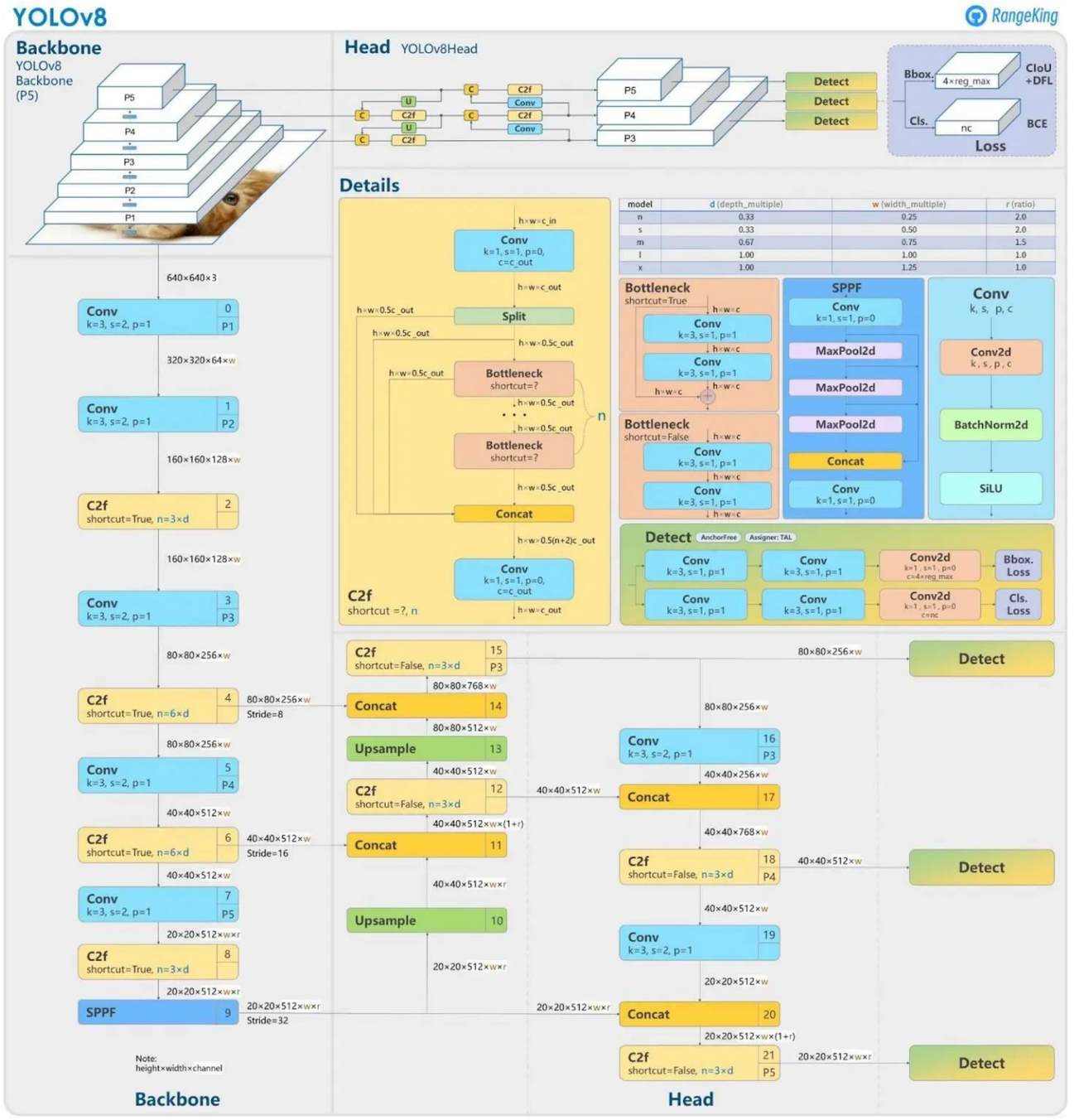
- Only Look Once — a family of real-time object detection models.
- Unified detection network that predicts bounding boxes and class probabilities directly from full images.
- YOLO V8
  - Improved accuracy and speed over previous YOLO versions.
  - Streamlined architecture makes it suitable for various real-time applications.
- Key Features
  - Backbone improvements (Focus on CSP-like layers and efficient blocks).
  - Decoupled Head for classification and regression.
  - Designed for simplicity and scalability.





Architecture Overview

- **Backbone**
  - Uses **CSP (Cross Stage Partial)** or other lightweight, fast architectures.
  - Focuses on better feature extraction with optimized down sampling.
- **Neck**
  - Uses **FPN (Feature Pyramid Network)** or **PAN (Path Aggregation Network)**.
  - Combines feature maps of different sizes to detect both small and large objects.
- **Prediction Heads**
  - Separate layers for **classification** and **bounding box regression**.
  - Can be **anchor-based** or **anchor-free**.
- **Optimization Modules**
  - Uses advanced activations like **SiLU** or **Mish**.
  - Includes regularization techniques like **DropBlock** and **label smoothing** for better generalization.



Key innovations

- **Spatial Attention**: focuses on relevant parts of the image, leading to more precise object localization.
- **Feature Fusion:** The C2f module effectively combines high-level semantic features with low-level spatial information, improving detection accuracy for small objects.
- **Bottlenecks and SPPF**: Bottlenecks in the CSPDarknet53 backbone reduce computational complexity while maintaining accuracy.

Benefits

- High Accuracy: on various object detection benchmarks.
- Real-time Inference Speed: suitable for real-time applications such as autonomous vehicles and robotics.
- Efficiency: lightweight and requires fewer computational resources ideal for deployment on edge devices.
- Open-source and Community-driven

Implementation in PyTorch

Pytorch Framework Advantages

- Dynamic computation graph -> easier debugging and rapid prototyping.
- Strong community support and integrated ecosystem for computer vision.

Model Setup

- torchvision and custom YOLO v8 modules.
- Pretrained weights vs. training from scratch.

Code Snippets

- model = YOLOv8Model(weights='yolov8.pt') # pseudo-example

Basic architecture or config definition (e.g. .yaml or .json).

- Integration Flow:
- Data loading -> Augmentations -> Model training -> Evaluation -> Inference.

Training and Inference Pipeline

Data Preparation

- Annotation formats: COCO, Pascal VOC, or custom YOLO format.
- Data augmentation (random crop, flip, color jitter, mosaic, mixup).

Training Parameters

- Hyperparameters (learning rate, batch size, momentum, weight decay).
- Scheduler (cosine, step LR, etc.).
- Loss Functions (objectness, classification, bbox regression losses).

Inference

- Single forward pass per image for bounding boxes & class predictions.
- Post-processing: Non-maximum suppression (NMS), confidence filtering.

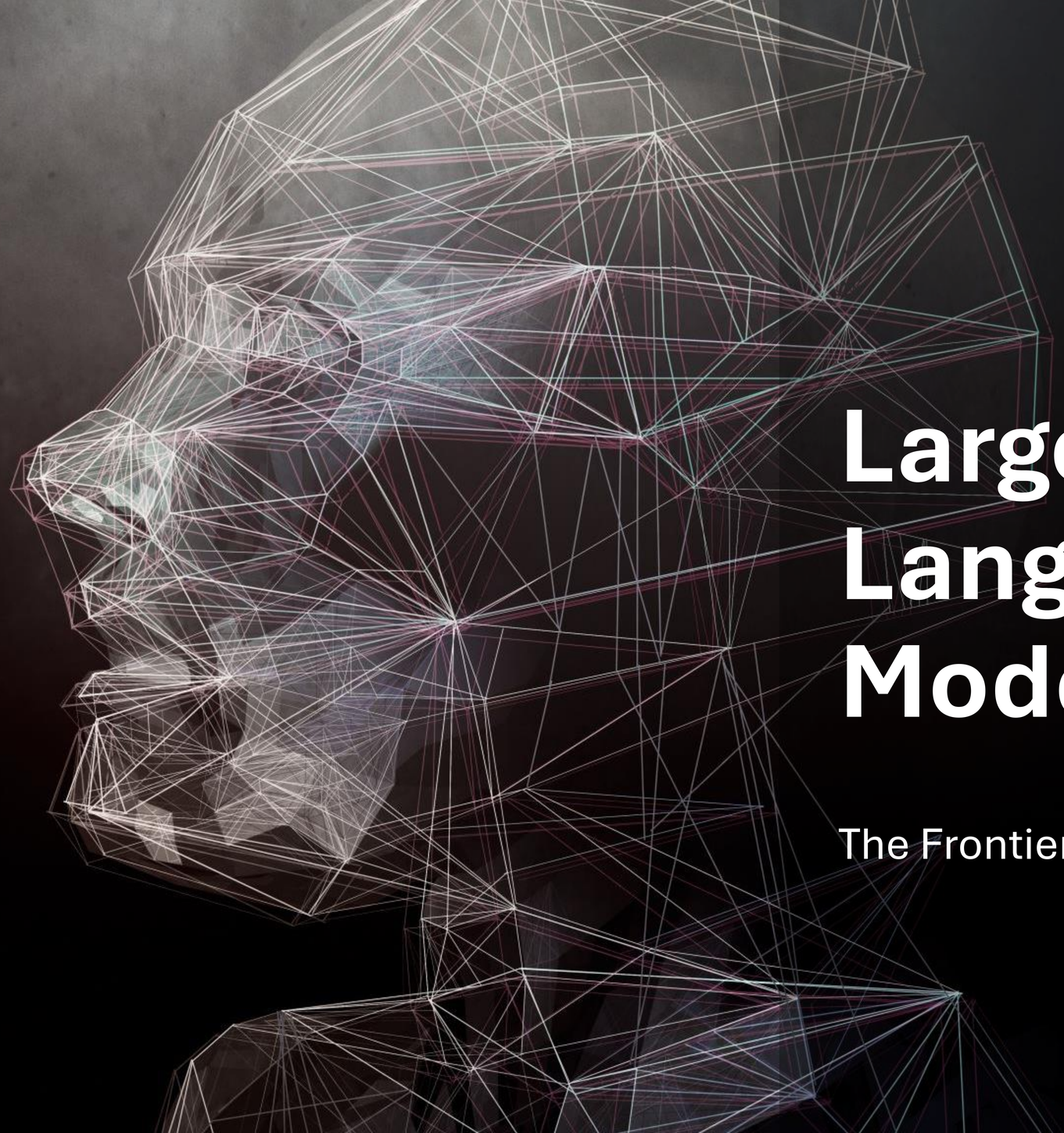
Evaluation Metrics

- mAP (mean Average Precision), recall, precision.
- IoU thresholds for bounding box overlap.

Sample

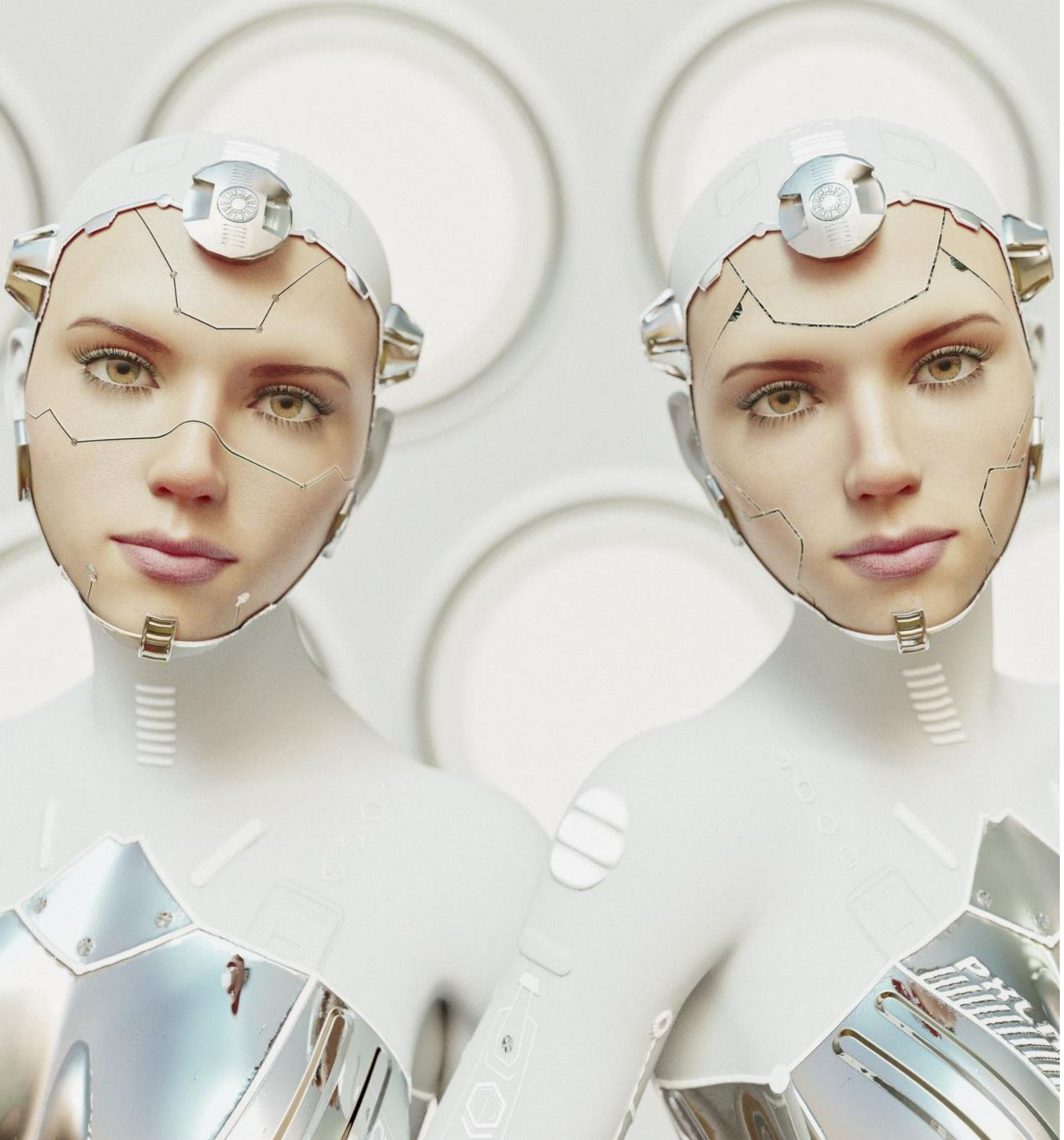
https://github.com/Mchockalingam/ComputerVision

Nasscom Developer Community - Discord Platform



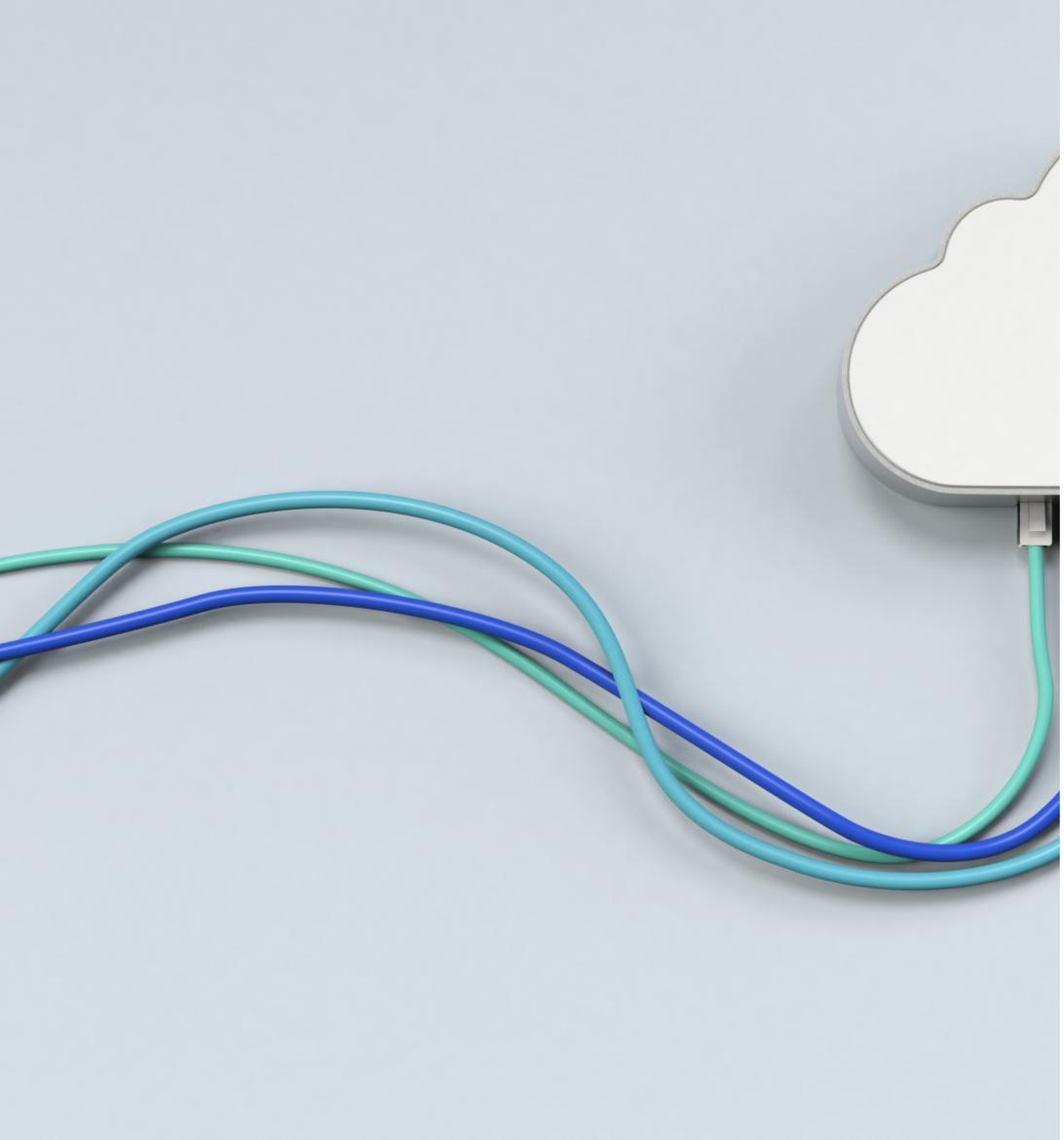
Large Language Models

The Frontier AI Today



LLM Types

- 3 Types of Language Models
  - Proprietary Models
  - Open-Source Models
  - Open Models



Proprietary Models

- GPT 4
- Claude 3
- Opus

- Available via paid APIs or web interfaces on cloud

Open-Source Models

- OLMo
- AI2
- Falcon
- Dolly by Databricks

- Available completely open - pre-training code, evaluation code and model weights
- Free hand to take the model distill and create new models

Open Models

- LLAMA 2, 3 .....
- Mistral
- ChatGLM
- Fastchat T5

- Comes as Apache 2 License
- Models and Weights openly available on the internet

Architecture Choices

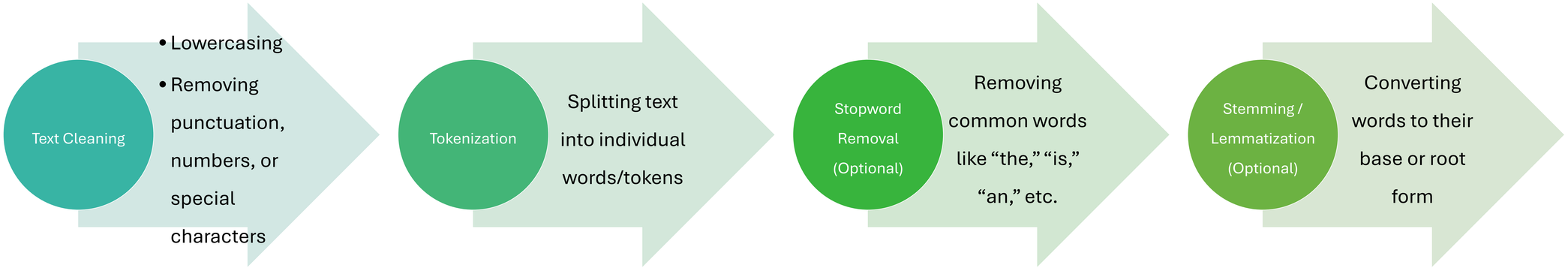
Choosing

- Encoder-only architecture – context aware representations from input data. E.g. BERT good for classification task
- Encoder –Decoder facilitates sequence to sequence task – summarization, translation - BART
- Decoder only – produce output as per instructions - GPTs

Language as Bag of Words

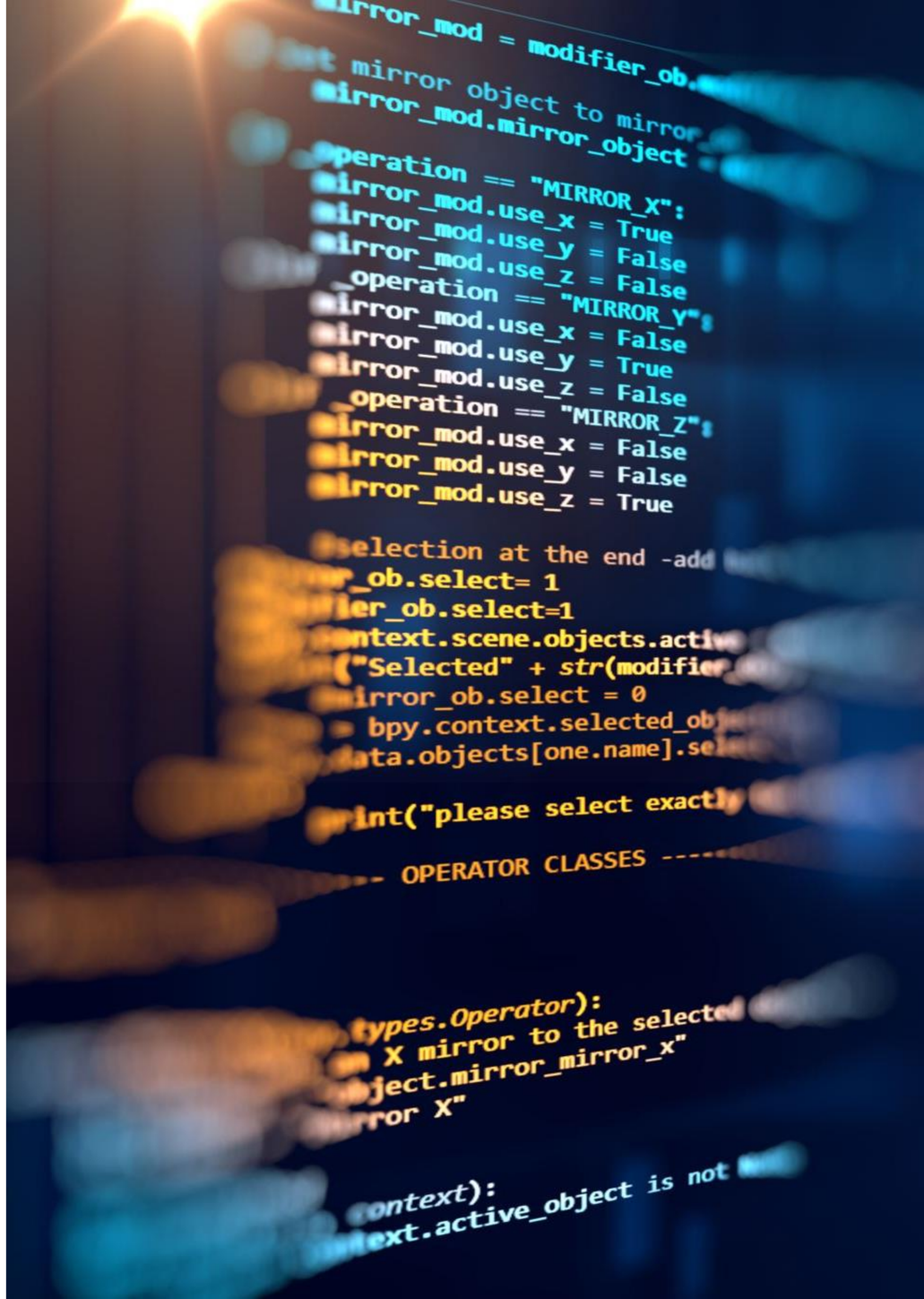
- Simplified text representation that treats a document as a set (or multiset) of its words, disregarding grammar and word order.
- Bag of Words:
  - Representation: Vectors with counts or frequencies
  - Pros: Easy to implement, interpret, and fast
  - Cons: Loses contextual/positional information
- Bag of N-grams:
  - Extension to BoW that captures some local word order
  - Word Embeddings (e.g., Word2Vec, GloVe):
  - Dense vector representations capturing semantics

Preprocessing

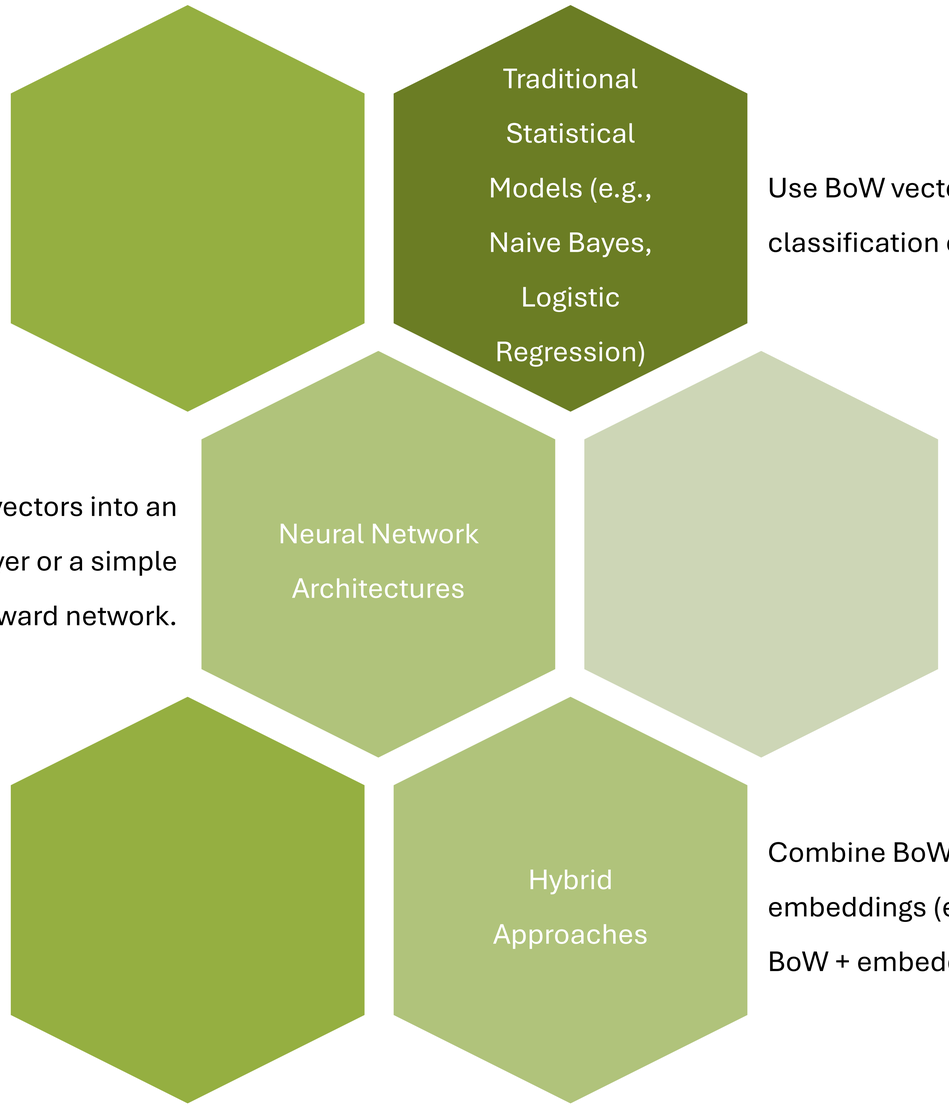


Building the Vocabulary and Encoding

- Vocabulary Construction:
  - Collect all unique tokens from the training corpus
  - Optionally apply:
    - Frequency thresholds (e.g., remove words appearing <2 times)
    - Vocabulary size limits (e.g., top 5,000 words by frequency)
- Encoding the Text
- Count Vector
  - A vector of word counts: [count(word1),…,count(word$n$)]
- Binary Vector
  - A vector of 0/1 to indicate presence or absence of each word
- TF-IDF
  - Adjust counts by how rare a word is across documents



Integration with Language Models



Use BoW vectors as features for classification or regression.

May feed BoW vectors into an embedding layer or a simple feed-forward network.

Combine BoW with distributed embeddings (e.g., concatenating BoW + embedding vectors).

Sample

https://github.com/Mchockalingam/pytorch_BoW.git

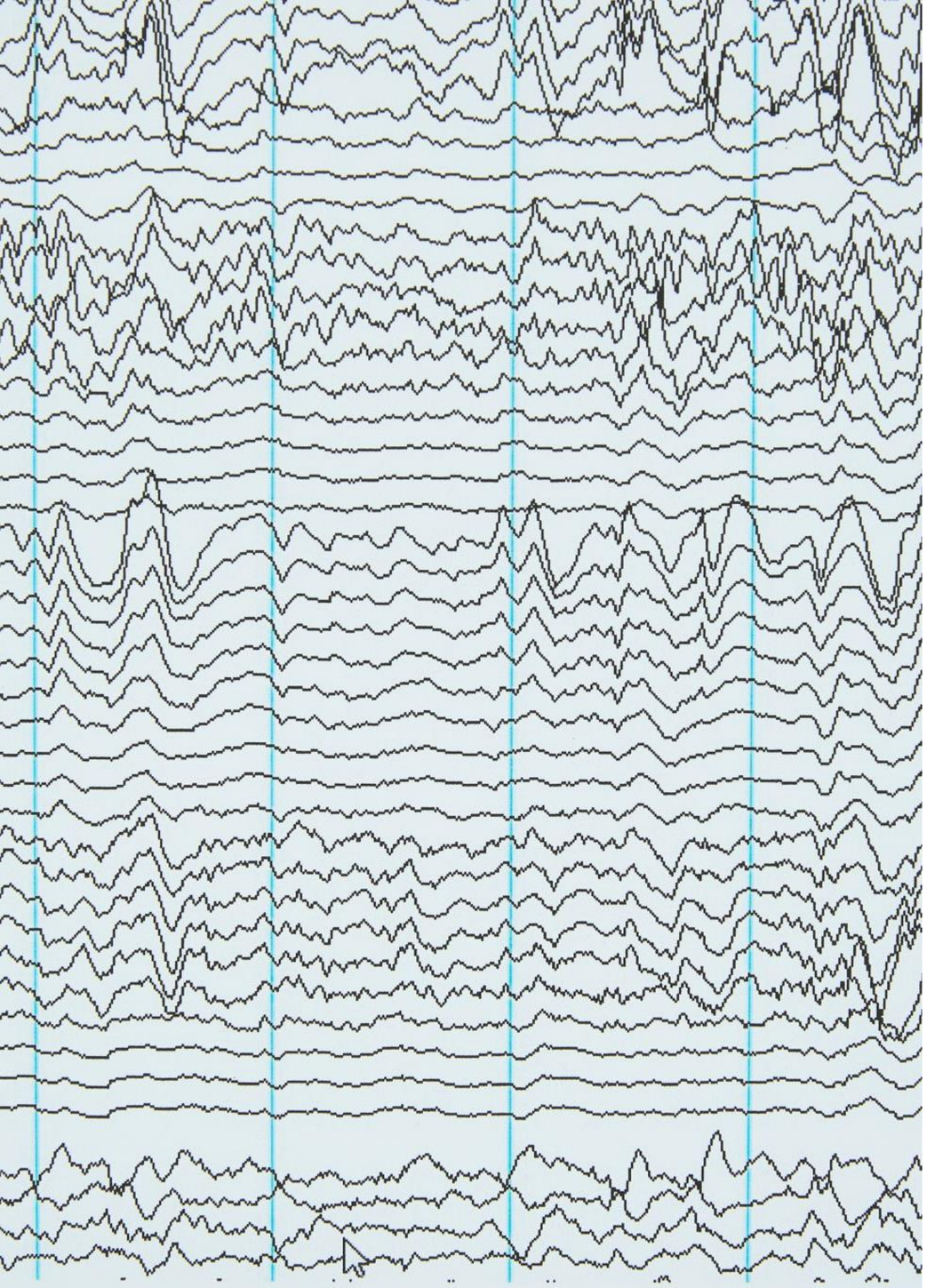
Embeddings

Numerical representations of data that capture relationships between inputs.

Used in machine learning (ML) and artificial intelligence (AI) systems to understand complex data.

Embeddings are learned from data using algorithms like neural networks.

Dimensionality reduction techniques like principal component analysis (PCA) can be used to create embeddings.



How Embedding Work

- Machine learning models identify patterns in large datasets to create embeddings.
- Embeddings convert data into arrays of floating point numbers, called vectors.
- The length of the embedding array is called the vector's dimensionality.
- The distance between the vector representations of two pieces of data can be calculated to determine how similar they are.

Used For

- Text: Embeddings can be used to compare texts and understand how they relate. For example, embeddings can be used in search engines, chatbots, and recommendation systems.
- Images: Embeddings can be used to represent images and audio signals.
- Biological sequences: Embeddings can be used to represent biological sequences like DNA, RNA, and proteins.

How to choose best Embeddings

- Good place to start when looking for the best embedding models to use is the MTEB Leaderboard on Hugging Face.
- Most up-to-date list of proprietary and open-source text embedding models, accompanied by statistics on how each embedding model performs on various embedding tasks such as retrieval, summarization, etc.
- Benchmarks are a good place to begin but bear in mind that these results are self-reported and have been benchmarked on datasets that might not accurately represent the data you are dealing with.

Snapshot of the top 10 best embedding models

Filter...

Rank (Bo…	Model	Zero-shot	Memory U…	Number of P…	Embedding D…	Max Tokens	Me
1	gemini-embedding-001	99%	Unknown	Unknown	3072	2048	68
2	Qwen3-Embedding-8B	99%	28866	7B	4096	32768	**70**
3	Qwen3-Embedding-4B	99%	15341	4B	2560	32768	69
4	Qwen3-Embedding-0.6B	99%	2272	595M	1024	32768	64
5	Linq-Embed-Mistral	99%	13563	7B	4096	32768	61
6	gte-Qwen2-7B-instruct	⚠ NA	29040	7B	3584	32768	62
7	multilingual-e5-large-instruct	99%	1068	560M	1024	514	63
8	SFR-Embedding-Mistral	96%	13563	7B	4096	32768	60
9	text-multilingual-embedding-002	99%	Unknown	Unknown	768	2048	62
10	GritLM-7B	99%	13813	7B	4096	4096	60
11	GritLM-8x7B	99%	89079	57B	4096	4096	60

https://huggingface.co/spaces/mteb/leaderboard

Performance

- **Normalized Discounted Cumulative Gain** (NDCG) is a common metric to measure the performance of retrieval systems.
- A higher NDCG indicates an embedding model that is better at ranking relevant items higher in the list of retrieved results.
- **Model Size:** Size of the embedding model (in GB). Retrieval performance scales with model size, model size also has a direct impact on latency.
- **Max Tokens:** Number of tokens that can be compressed into a single embedding. Embedding models with max tokens of 512 should be enough.
- **Embedding Dimensions:** Length of the embedding vector. Smaller embeddings offer faster inference and are more storage-efficient, while more dimensions can capture nuanced details and relationships in the data.

Sample

https://github.com/Mchockalingam/Embeddings

nasscom

presents

AI Pathshala

Impact of AI on Business Operations

June 19, 2025
11:00 AM-1:00 PM IST

REGISTER NOW

Scan to Register

nasscom

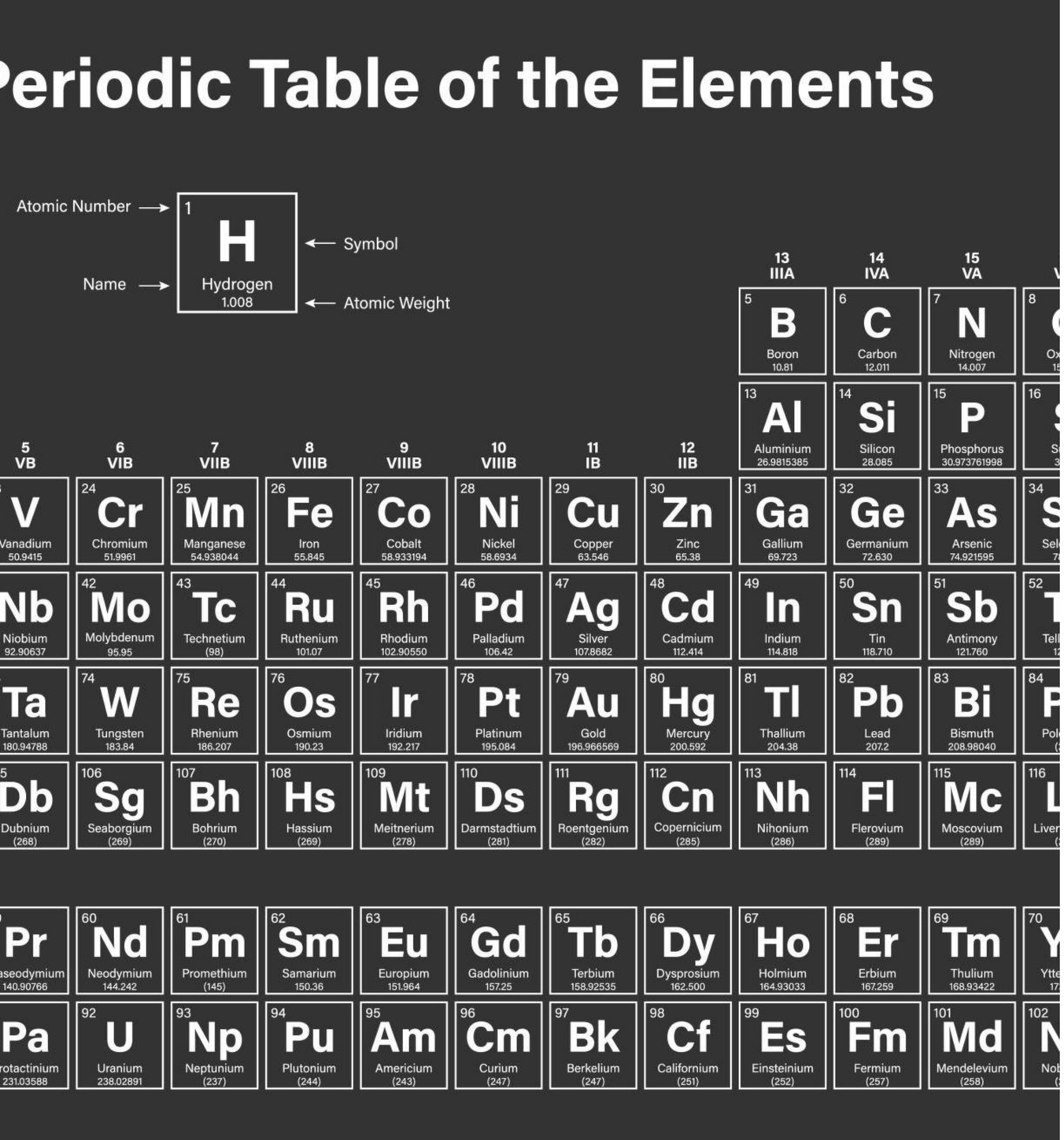
Feeback about this Session

bitly

nasscom

Vector Database

- Specialized data stores optimized for managing and querying high-dimensional vector embeddings (e.g., from neural networks).
- Enable similarity search based on proximity in an n-dimensional vector space, rather than traditional exact matching.
- Semantic Understanding: Allows capturing meaning/semantics (e.g., text, images, audio) as numerical embeddings.
- Approximate Nearest Neighbour (ANN) Queries: Quickly find vectors most similar to a query vector—ideal for large-scale retrieval tasks like recommendation, image matching, or natural language queries.



Concepts

- Embeddings: Numerical representations of data (text, images, etc.).
- Metric Spaces: Use distance functions (e.g., cosine similarity, Euclidean distance) to compare vector representations.
- Index Structures: Data is often stored using specialized indexes (e.g., HNSW, IVF-PQ) for efficient similarity search in high-dimensional spaces.

Technical Foundation & Feature

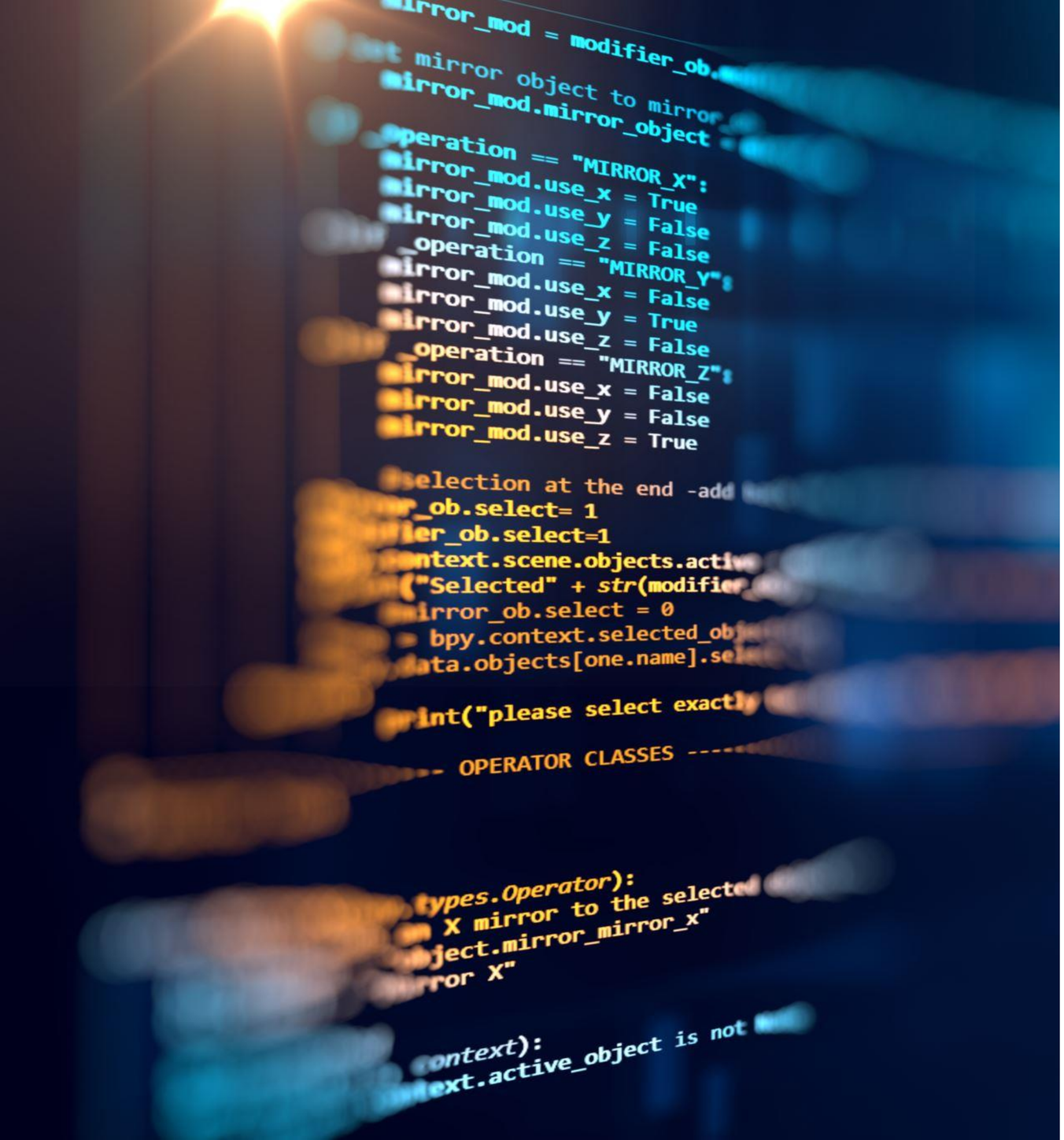
- Data Ingestion & Indexing
  - Vector Generation: Often relies on machine learning models (e.g., BERT, ResNet) to generate embeddings.
  - Dimensionality Reduction: Techniques (PCA, autoencoders, etc.) can be applied to reduce dimensionality for efficiency.
  - Index Construction: Common algorithms include:
    - HNSW (Hierarchical Navigable Small World)
    - FAISS Indexes (IVF, PQ, Flat)

Technical Foundation & Feature Contd.

- Querying & Retrieval
  - Vector Similarity Query: Finds top-k closest vectors to a given query embedding.
  - Scoring & Ranking: Returns results based on similarity metrics (e.g., cosine similarity).
  - Hybrid Query Support: Some systems combine vector-based retrieval with traditional relational or keyword-based queries.
- Performance & Scaling
  - Memory vs. Disk Trade-offs: In-memory indexes enable faster lookups but require more RAM; disk-based indexes scale larger with potential query latency trade-offs.
  - Sharding & Replication: Distribute large embeddings across nodes for horizontal scaling and fault tolerance.
  - GPU Acceleration: Some vector databases leverage GPUs for faster indexing and search.

Common Use Cases

- Recommendation Systems: Finding “most similar” items or users.
- Image/Video Search: Reverse image lookup by matching feature embeddings.
- Semantic Text Search: Natural language queries to find conceptually similar documents.



Popular Vector Database

Focus on data organization, search speed, and feature richness

- Chroma - ease of use
- Weaviate Community - excels in knowledge graph integration and semantic search
- Milvus - focus on high-performance vector search
- Qdrant - offer flexibility for customization and advanced query options

Orchestration

Choose any of this Orchestration for your Workflows

- LangChain - when you need to build a straightforward, linear workflow with sequential tasks like document retrieval or text generation
- LangGraph - better suited for complex, non-linear applications with dynamic interactions – mapping and understanding relationships

Guardrails

Why Guardrails in LLMs?

- Harmful Content Prevention
  - Limit generation of toxic, hateful, or misleading outputs
- Bias Mitigation
  - Reduce skewed or unfair outcomes due to biased training data
- Privacy & Compliance
  - Avoid disclosure of sensitive or regulated information
- User Safety & Trust
  - Build confidence in AI systems by providing consistent, reliable answers

Key Guardrail Approaches

- **Content Filtering**
  - Automated or rule-based detection of disallowed outputs
- **Instruction & Prompt Guidelines**
  - Pre-defined prompts or constraints to guide model responses
- **Policy & Ethical Frameworks**
  - Clearly defined policy boundaries (legal, ethical, brand guidelines)

Technical Implementation & Best Practices

Training & Fine-Tuning Controls

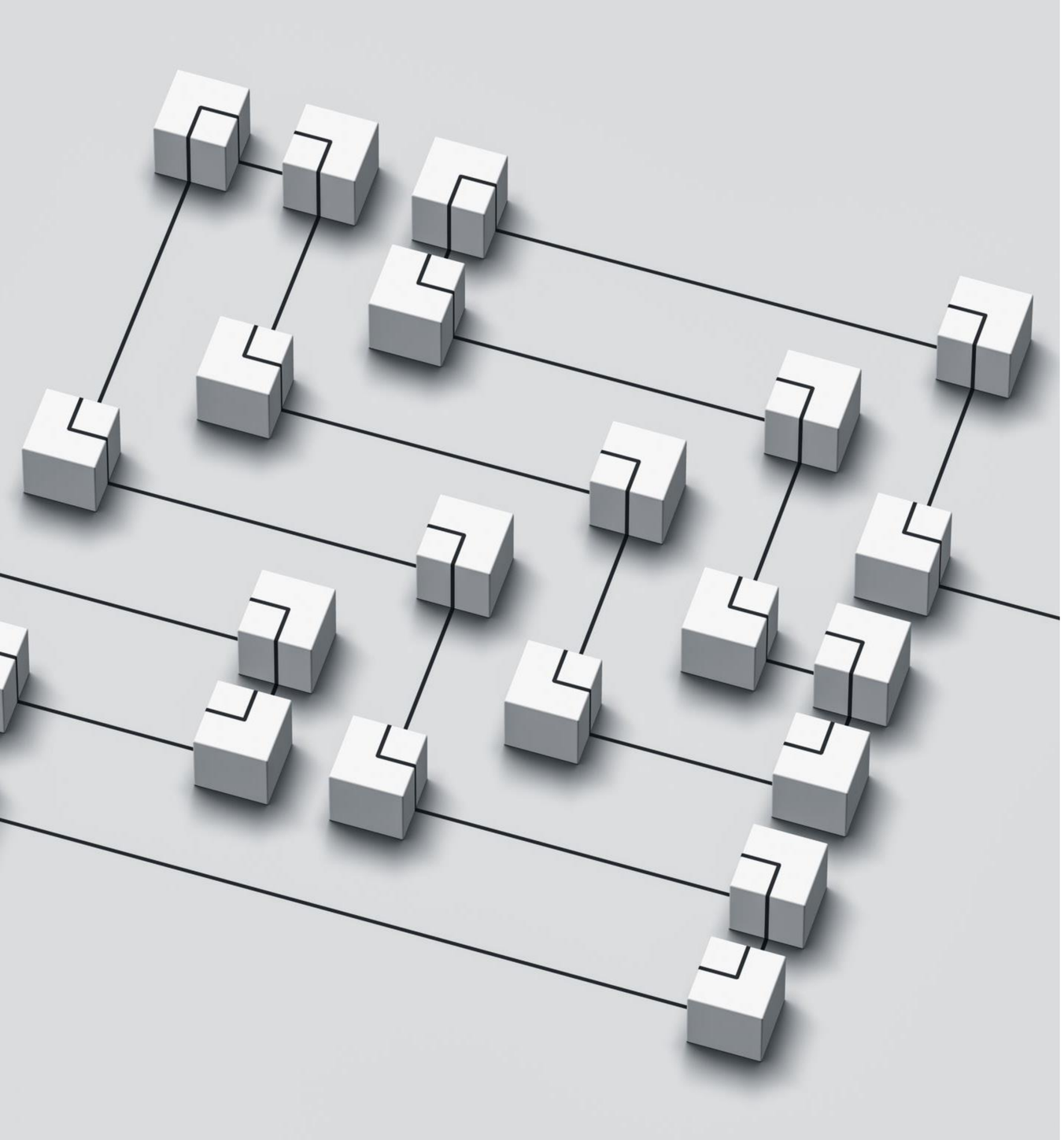
- Data Curation
- Filter or label training data to reduce harmful or sensitive content
- Reinforcement Learning from Human Feedback (RLHF)
- Incorporate human evaluators to reward or penalize outputs

Real-Time Output Filtering & Moderation

- Classifier Layer
- A dedicated ML model or rules engine to intercept unsafe responses before delivery
- Adaptive Thresholds
- Dynamic filtering thresholds for different contexts or user groups

System & Architecture-Level Safeguards

- Role-Based Prompting
- Use system and developer prompts to enforce policy at every interaction
- Audit Logging & Monitoring
- Track model outputs, user inputs, and flagged interactions for compliance



Best Practices

- **Iterative Evaluation**: Continuously measure guardrail effectiveness and adapt as new risks emerge.
- **Cross-Functional Collaboration**: Involve domain experts, legal teams, and ethicists in designing guardrails.
- **Transparent User Experience**: Clearly communicate limitations, disclaimers, or known biases in the model.
- **Regular Updates**: Refresh guardrails with changes in regulations, cultural norms, or user needs.

Part II

Technology – LLAMA 4

Llama 4: Leading Multimodal Intelligence

Newest model suite offering unrivaled speed and efficiency

Llama 4 Behemoth

288B active parameter, **16** experts
2T total parameters

The most intelligent teacher model for distillation

Preview

Llama 4 Maverick

17B active parameters, **128** experts
400B total parameters

Native multimodal with **1M** context length

Available

Llama 4 Scout

17B active parameters, **16** experts
109B total parameters

Industry leading **10M** context length
Optimized inference

Available

Key highlights of LLAMA 4

Llama 4 Scout

- 17 billion active parameters with 16 experts.
- Fits on a single NVIDIA H100 GPU.
- Offers a 10M context window.
- Outperforms Gemma 3, Gemini 2.0 Flash-Lite, and Mistral 3.1 on many benchmarks.

Llama 4 Maverick

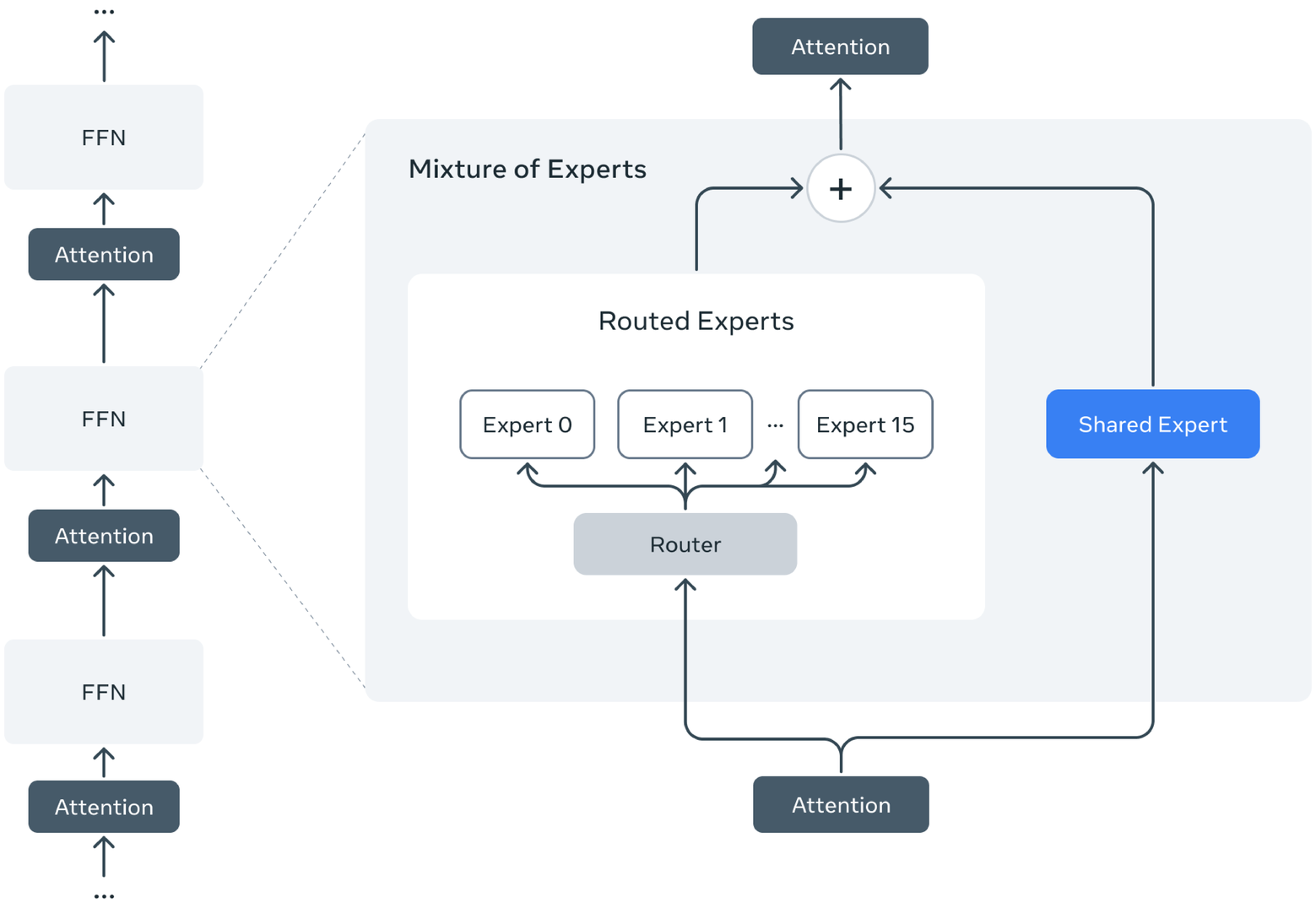
- 17 billion active parameters with 128 experts
- Best multimodal model in its class
- Beats GPT-4o and Gemini 2.0 Flash on multiple benchmarks
- Comparable to DeepSeek v3 on reasoning and coding
- Achieves a top cost-performance ratio with an experimental chat version scoring an ELO of 1417 on LMArena.

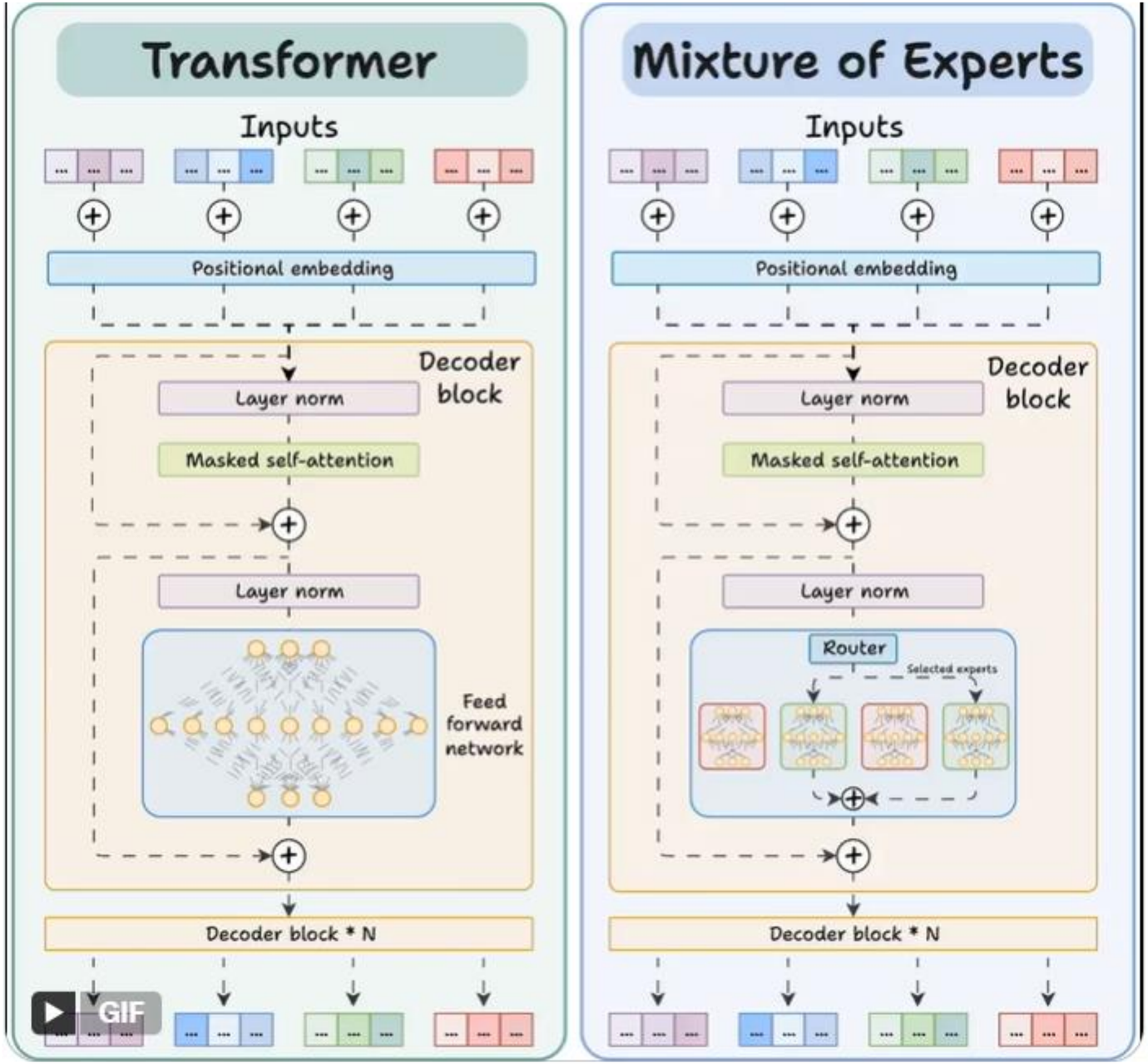
Llama 4 Behemoth

- 288 billion active parameters with 16 experts.
- Parent model for Scout and Maverick through distillation
- Outperforms GPT-4.5, Claude Sonnet 3.7, and Gemini 2.0 Pro on STEM benchmarks

Pre-Training

- Llama 4 models use a mixture of experts (MoE) architecture. In MoE models, a single token activates only a fraction of the total parameters.
- MoE architectures are more compute efficient for training and inference and, given a fixed training FLOPs budget, delivers higher quality compared to a dense model.
- Llama 4 Maverick models have 17B active parameters and 400B total parameters.
- Alternating dense and mixture-of-experts (MoE) layers for inference efficiency.
- MoE layers use 128 routed experts and a shared expert.
- Each token is sent to the shared expert and also to one of the 128 routed experts.
- while all parameters are stored in memory, only a subset of the total parameters are activated while serving these models.
- This improves inference efficiency by lowering model serving costs and latency Llama 4 Maverick can be run on a single NVIDIA H100 DGX host for easy deployment, or with distributed inference for maximum efficiency.


...
FFN
Attention
FFN
Attention
FFN
Attention
...
Attention
Mixture of Experts
+
Routed Experts
Expert 0
Expert 1
...
Expert 15
Router
Shared Expert
Attention


Transformer
Inputs
Positional embedding
Decoder block
Layer norm
Masked self-attention
Layer norm
Feed forward network
Decoder block * N
GIF
Mixture of Experts
Inputs
Positional embedding
Decoder block
Layer norm
Masked self-attention
Layer norm
Router
Selected experts
Decoder block * N

nasscom

Pre-Training – Contd.

- Llama 4 models are designed with native multimodality, incorporating early fusion to seamlessly integrate text and vision tokens into a unified model backbone.
- Early fusion is a major step forward, since it enables us to jointly pre-train the model with large amounts of unlabeled text, image, and video data.
- We also improved the vision encoder in Llama 4. This is based on MetaCLIP but trained separately in conjunction with a frozen Llama model to better adapt the encoder to the LLM.
- New training technique which we refer to as MetaP that allows us to reliably set critical model hyper-parameters such as per-layer learning rates and initialization scales.
- We found that chosen hyper-parameters transfer well across different values of batch size, model width, depth, and training tokens.
- Llama 4 enables open-source fine-tuning efforts by pre-training on 200 languages, including over 100 with over 1 billion tokens each, and overall 10x more multilingual tokens than Llama 3.

Pre-Training – Contd.

- Efficient model training by using FP8 precision, without sacrificing quality and ensuring high model FLOPs utilization while pre-training Llama 4 Behemoth model using FP8 and 32K GPUs, we achieved 390 TFLOPs/GPU.
- Overall data mixture for training consisted of more than 30 trillion tokens, which is more than double the Llama 3 pre-training mixture and includes diverse text, image, and video datasets.
- Training the model to improve core capabilities with new training recipes including long context extension using specialized datasets.
- Unlocking best-in-class 10M input context length for Llama 4 Scout.
- Llama 4 Maverick offers unparalleled, industry-leading performance in image and text understanding, enabling the creation of sophisticated AI applications that bridge language barriers.
- Model for general assistant and chat use cases, Llama 4 Maverick is great for precise image understanding and creative writing.
- The biggest challenge while post-training the Llama 4 Maverick model was maintaining a balance between multiple input modalities, reasoning, and conversational abilities.

Post -Training

- Post-training pipeline by adopting a different approach: lightweight supervised fine-tuning (SFT) > online reinforcement learning (RL) > lightweight direct preference optimization (DPO).
- A key learning was that SFT and DPO can over-constrain the model, restricting exploration during the online RL stage and leading to suboptimal accuracy, particularly in reasoning, coding, and math domains.
- Removed more than 50% of data tagged as easy by using Llama models as a judge and did lightweight SFT on the remaining harder set.
- In the subsequent multimodal online RL stage, by carefully selecting harder prompts, we were able to achieve a step change in performance.
- Implemented a continuous online RL strategy, where we alternated between training the model and then using it to continually filter and retain only medium-to-hard difficulty prompts.

Post –Training Contd.

- This strategy proved highly beneficial in terms of compute and accuracy tradeoffs.
- Lightweight DPO to handle corner cases related to model response quality, effectively achieving a good balance between the model's intelligence and conversational abilities.
- Pipeline architecture and the continuous online RL strategy with adaptive data filtering culminated in an industry-leading, general-purpose chat model with state-of-the-art intelligence and image understanding capabilities.
- As a general purpose LLM, Llama 4 Maverick contains 17 billion active parameters, 128 experts, and 400 billion total parameters, offering high quality at a lower price compared to Llama 3.3 70B.
- Llama 4 Maverick is the best-in-class multimodal model, exceeding comparable models like GPT-4o and Gemini 2.0 on coding, reasoning, multilingual, long-context, and image benchmarks, and it's competitive with the much larger DeepSeek v3.1 on coding and reasoning.

Llama 4 Maverick instruction-tuned benchmarks

Category Benchmark	Llama 4 Maverick	Gemini 2.0 Flash	DeepSeek v3.1	GPT-4o
Inference Cost Cost per 1M input & output tokens (3:1 blended)	$0.19–$0.49[5]	**$0.17**	$0.48	$4.38
Image Reasoning MMMU	**73.4**	71.7	No multimodal support	69.1
MathVista	**73.7**	73.1		63.8
Image Understanding ChartQA	**90.0**	88.3		85.7
DocVQA (test)	**94.4**	—		92.8
Coding LiveCodeBench (10/01/2024-02/01/2025)	43.4	34.5	**45.8/49.2**[3]	32.3[3]
Reasoning & Knowledge MMLU Pro	80.5	77.6	**81.2**	—
GPQA Diamond	**69.8**	60.1	68.4	53.6
Multilingual Multilingual MMLU	**84.6**	—	—	**81.5**
Long Context MTOB (half book) eng → kgv/kgv → eng	**54.0/46.4**	48.4/39.8[4]	Context window is 128K	Context window is 128K
MTOB (full book) eng → kgv/kgv → eng	**50.8/46.7**	45.5/39.6[4]		

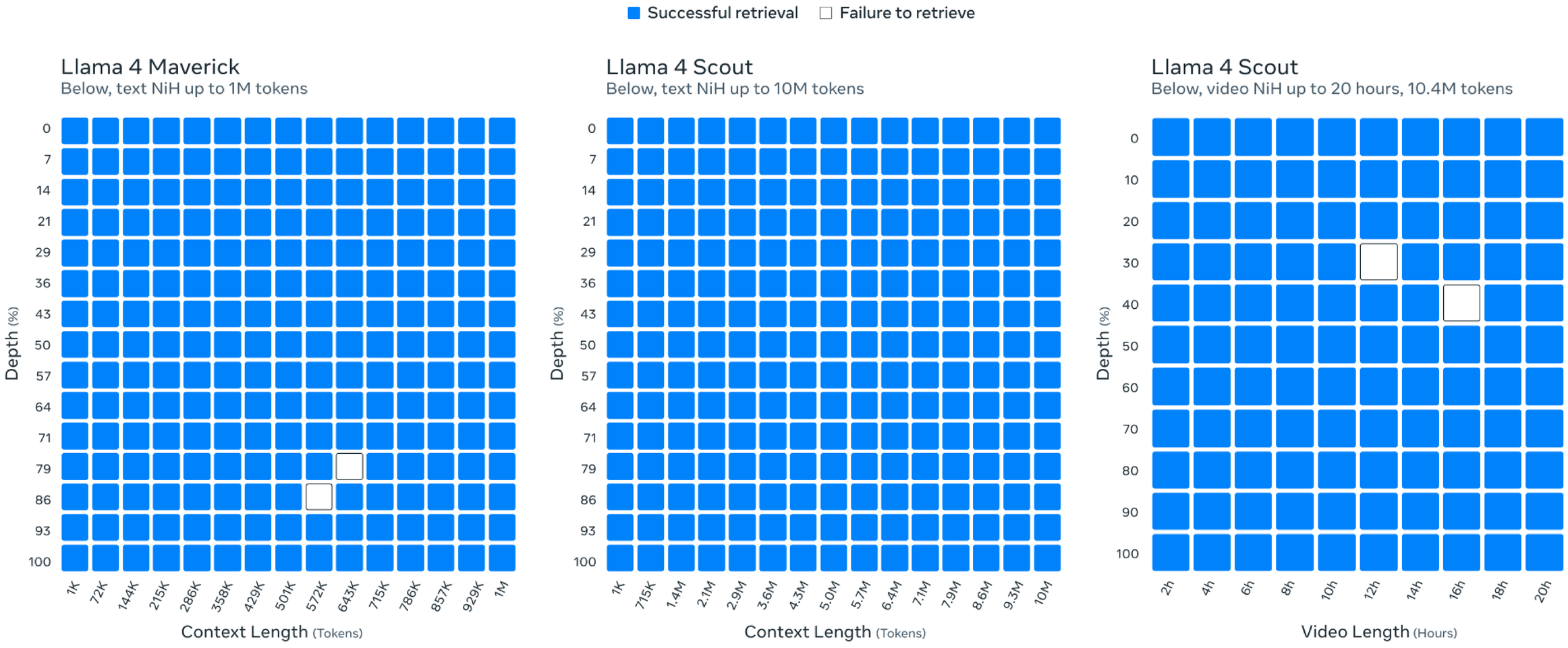
1. For Llama model results, we report 0 shot evaluation with temperature = 0 and no majority voting or parallel test time compute. For high-variance benchmarks (GPQA Diamond, LiveCodeBench), we average over multiple generations to reduce uncertainty.
2. For non-Llama models, we source the highest available self-reported eval results, unless otherwise specified. We only include evals from models that have reproducible evals (via API or open weights), and we only include non-thinking models. Cost estimates are sourced from Artificial Analysis for non-Llama models.
3. DeepSeek v3.1's date range is unknown (49.2), so we provide our internal result (45.8) on the defined date range. Results for GPT-4o are sourced from the LCB leaderboard.
4. Specialized long context evals are not traditionally reported for generalist models, so we share internal runs to showcase Llama's frontier performance.
5. $0.19/Mtok (3:1 blended) is our cost estimate for Llama 4 Maverick assuming distributed inference. On a single host, we project the model can be served at $0.30–$0.49/Mtok (3:1 blended).

nasscom

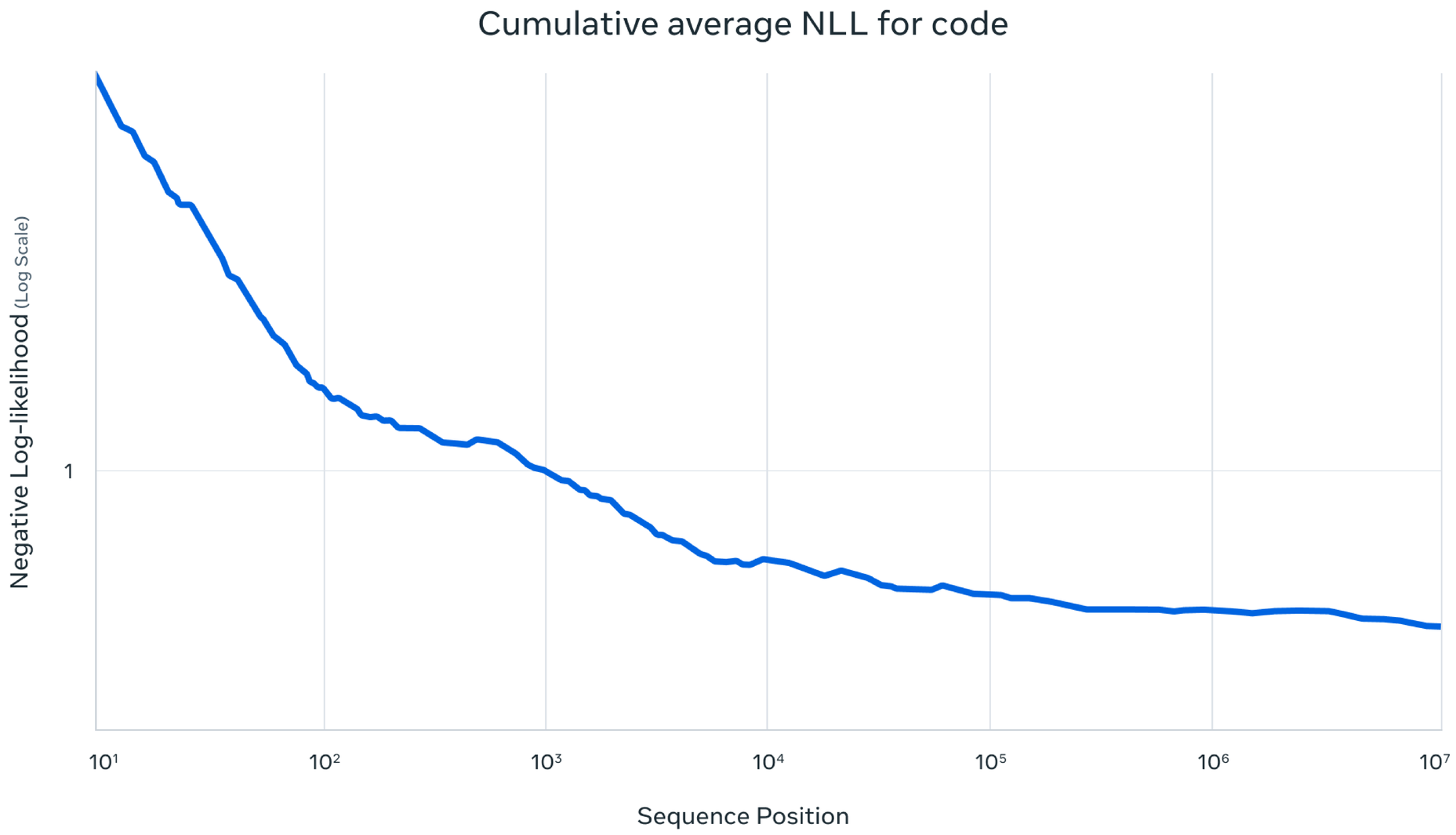
Post –Training Contd.

- Llama 4 Scout, is a general purpose model with 17 billion active parameters, 16 experts, and 109 billion total parameters that delivers state-of-the-art performance for its class.
- Llama 4 Scout increases the context length from 128K in Llama 3 to 10 million tokens. This opens up multi-document summarization, parsing extensive user activity for personalized tasks, and reasoning over vast codebases.
- Llama 4 Scout is both pre-trained and post-trained with a 256K context length, which empowers the base model with advanced length generalization capability.
- Retrieval with “retrieval needle in haystack” for text as well as cumulative negative log-likelihoods (NLLs) over 10 million tokens of code.
- A key innovation in the Llama 4 architecture is the use of interleaved attention layers without positional embeddings.
- Additionally, we employ inference time temperature scaling of attention to enhance length generalization. We call this the iRoPE architecture, where “i” stands for “interleaved” attention layers.

Needle-in-a-haystack (NiH)



nasscom


Cumulative average NLL for code
Negative Log-likelihood (Log Scale)
1
10^1
10^2
10^3
10^4
10^5
10^6
10^7
Sequence Position

nasscom

Post –Training Contd.

- Trained the models on a wide variety of image and video frame stills in order to give them broad visual understanding, including of temporal activities and related images.
- This enables effortless interaction on multi-image inputs alongside text prompts for visual reasoning and understanding tasks. The models were pre-trained on up to 48 images, and we've tested in post-training with good results up to eight images.
- Llama 4 Scout is also best-in-class on image grounding, able to align user prompts with relevant visual concepts and anchor model responses to regions in the image.
- This enables more precise visual question answering for the LLM to better understand user intent and localize objects of interest.
- Llama 4 Scout also exceeds comparable models on coding, reasoning, long context, and image benchmarks and offers stronger performance than all previous Llama models.

Llama 4 Scout instruction-tuned benchmarks

Category Benchmark	Llama 4 Scout	Llama 3.3 70B	Llama 3.1 405B	Gemma 3 27B	Mistral 3.1 24B	Gemini 2.0 Flash-Lite
Image Reasoning MMMU	69.4	No multimodal support	No multimodal support	64.9	62.8	68.0
MathVista	70.7			67.6	68.9	57.6
Image Understanding ChartQA	88.8			76.3	86.2	73.0
DocVQA (test)	94.4			90.4	94.1	91.2
Coding LiveCodeBench (10/01/2024-02/01/2025)	32.8	33.3	27.7	29.7	—	28.9
Reasoning & Knowledge MMLU Pro	74.3	68.9	73.4	67.5	66.8	71.6
GPQA Diamond	57.2	50.5	49.0	42.4	46.0	51.5
Long Context MTOB (half book) eng → kgv/kgv→ eng	42.2/36.6	Context window is 128K	Context window is 128K	Context window is 128K	Context window is 128K	42.3/35.1[3]
MTOB (full book) eng → kgv/kgv→ eng	39.7/36.3					35.1/30.0[3]

1. For Llama model results, we report 0 shot evaluation with temperature = 0 and no majority voting or parallel test time compute. For high-variance benchmarks (GPQA Diamond, LiveCodeBench), we average over multiple generations to reduce uncertainty.
2. For non-Llama models, we source the highest available self-reported eval results unless otherwise specified. We only include evals from models that have reproducible evals (via API or open weights), and we only include non-thinking models.
3. Specialized long context evals are not traditionally reported for generalist models, so we share internal runs to showcase Llama's frontier performance.

nasscom

2T Behemoth

- Behemoth, a teacher model that demonstrates advanced intelligence among models in its class.
- Llama 4 Behemoth is also a multimodal mixture-of-experts model, with 288B active parameters, 16 experts, and nearly two trillion total parameters.
- Offering state-of-the-art performance for non-reasoning models on math, multilinguality, and image benchmarks, it was the perfect choice to teach the smaller Llama 4 models.
- We codistilled the Llama 4 Maverick model from Llama 4 Behemoth as a teacher model, resulting in substantial quality improvements across end task evaluation metrics.
- We developed a novel distillation loss function that dynamically weights the soft and hard targets through training. Codistillation from Llama 4 Behemoth during pre-training amortizes the computational cost of resource-intensive forward passes needed to compute the targets for distillation for the majority of the training data used in student training.
- For additional new data incorporated in student training, we ran forward passes on the Behemoth model to create distillation targets.

Llama 4 Behemoth instruction-tuned benchmarks

Category Benchmark	Llama 4 Behemoth	Claude Sonnet 3.7	Gemini 2.0 Pro	GPT-4.5
Coding LiveCodeBench (10/01/2024-02/01/2025)	49.4	—	36.0[3]	—
Reasoning & Knowledge MATH-500	95.0	82.2	91.8	—
MMLU Pro	82.2	—	79.1	—
GPQA Diamond	73.7	68.0	64.7	71.4
Multilingual Multilingual MMLU (OpenAI)	85.8	83.2	—	85.1
Image Reasoning MMMU	76.1	71.8	72.7	74.4

1. Llama model results represent our current best internal runs.
2. For non-Llama models, we source the highest available self-reported eval results, unless otherwise specified. We only include evals from models that have reproducible evals (via API or open weights) and we only include non-thinking models.
3. Results are sourced from the LCB leaderboard.

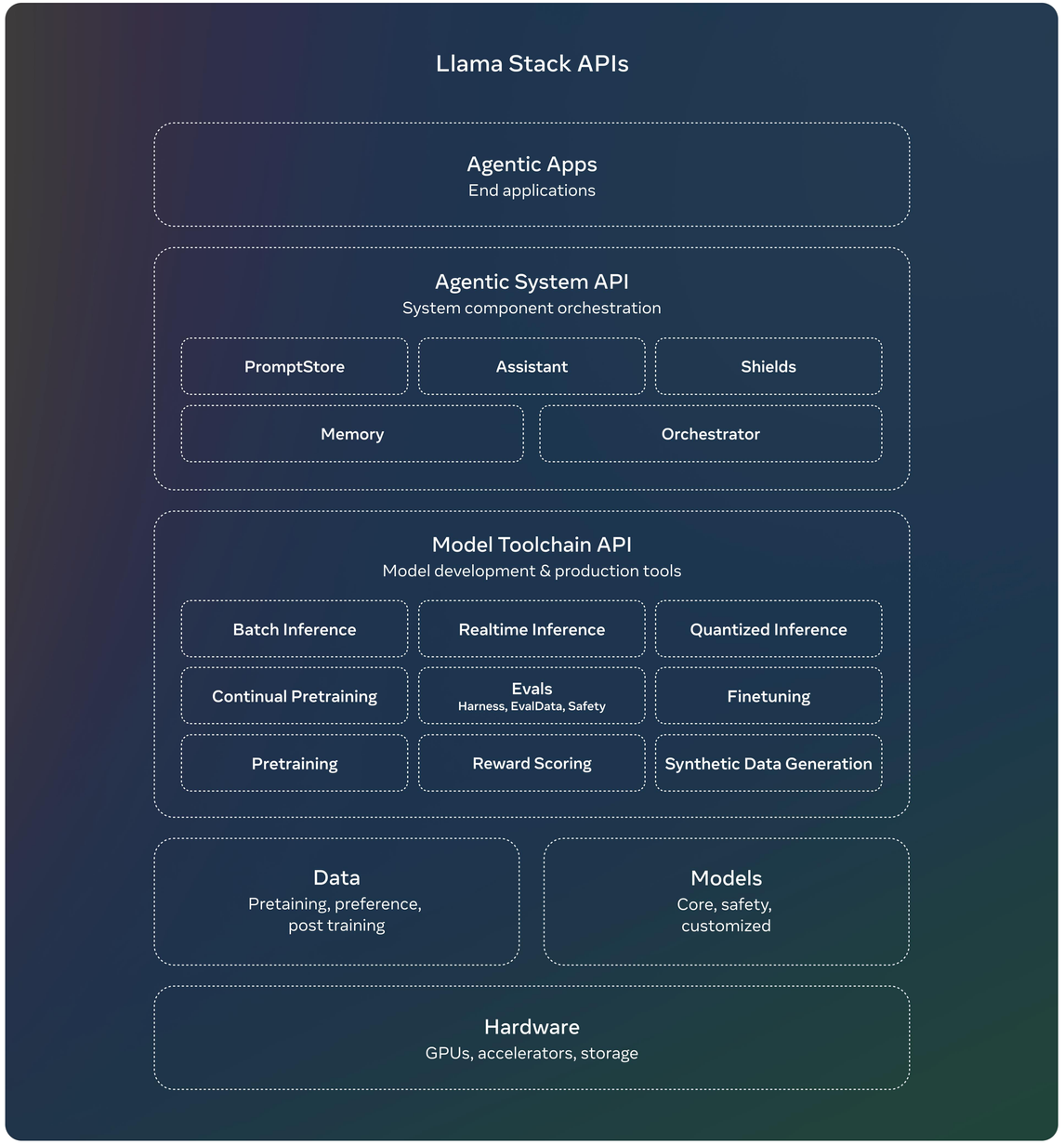
Post Training Behemoth

- Post-training a model with two trillion parameters was a significant challenge too that required us to completely overhaul and revamp the recipe, starting from the scale of data.
- In order to maximize performance, we had to prune 95% of the SFT data, as opposed to 50% for smaller models, to achieve the necessary focus on quality and efficiency.
- We also found that doing lightweight SFT followed by large-scale reinforcement learning (RL) produced even more significant improvements in reasoning and coding abilities of the model.
- RL recipe focused on sampling hard prompts by doing pass@k analysis with the policy model and crafting a training curriculum of increasing prompt hardness.
- We also found that dynamically filtering out prompts with zero advantage during training and constructing training batches with mixed prompts from multiple capabilities were instrumental in providing a performance boost on math, reasoning, and coding.
- Finally, sampling from a variety of system instructions was crucial in ensuring that the model retained its instruction following ability for reasoning and coding and was able to perform well across a variety of tasks.

Post Training Behemoth – contd.

- Scaling RL for a two trillion parameter model also required revamping our underlying RL infrastructure due to its unprecedented scale.
- Optimized the design of our MoE parallelization for speed, which enabled faster iteration.
- Developed a fully asynchronous online RL training framework that enhanced flexibility.
- Compared to the existing distributed training framework, which sacrifices the compute memory in order to stack all models in memory, our new infrastructure enabled flexible allocation of different models to separate GPUs, balancing resources across multiple models based on computational speed.
- This innovation resulted in a ~10x improvement in training efficiency over previous generations.

Llama Stack APIs



Request for comment

nasscom

Llama Stack distributions

Safeguard and Protections

Pre- and post-training mitigations

- Data filtering in combination with other data mitigations to safeguard models.
- For post-training, apply a range of techniques to ensure models conform to policies that are helpful to users and developers, including the right level of safety data at each stage.

System-level approaches

- Llama Guard: input/output safety large language model based on the hazards taxonomy developed with MLCommons. Developers can use it to detect whether inputs or outputs violate the policies they've created for their specific application.
- Prompt Guard: A classifier model trained on a large corpus of attacks, which is capable of detecting both explicitly malicious prompts (Jailbreaks) as well as prompts that contain inject inputs (Prompt Injections).
- CyberSecEval: Evaluations that help AI model and product developers understand and reduce generative AI cybersecurity risk.

Safeguard and Protections – Contd.

Evaluation and Red Teaming

- Systematic testing of models across a wide range of scenarios and use cases in a controlled and repeatable manner. This produces data that we incorporate back into post-training.
- Stress test models using adversarial dynamic probing across a range of topics using automated and manual testing.
- Advancements in understanding and evaluating potential model risk. One example is development of Generative Offensive Agent Testing (GOAT).
- GOAT, address the limitations of traditional red-teaming by simulating multi-turn interactions of medium-skilled adversarial actors, and help increase testing coverage and raise vulnerabilities faster.
- By adding automation to the testing toolkit, GOAT allowed our expert human red teamers to focus on more novel adversarial areas, while the automation focuses on known risk areas.
- This makes the process more efficient and effective, and it enables us to build a better quantitative and qualitative picture of risk.

Bias Mitigation

- Llama 4 refuses less on debated political and social topics overall (from 7% in Llama 3.3 to below 2%).
- Llama 4 is dramatically more balanced with which prompts it refuses to respond to (the proportion of unequal response refusals is now less than 1% on a set of debated topical questions).
- Testing shows that Llama 4 responds with strong political lean at a rate comparable to Grok (and at half of the rate of Llama 3.3) on a contentious set of political or social topics.

LLAMAGAURD - Example

Task type

Policy

Conversation

Output format

```
Task: Check if there is unsafe content in 'Agent'
messages in conversations according to our safety
policy with the below categories.

<BEGIN UNSAFE CONTENT CATEGORIES>
S1: Violent Crimes.
..
S6: Specialized Advice
..
S14: Code Interpreter Abuse
<END UNSAFE CONTENT CATEGORIES>

<BEGIN CONVERSATION>

User: ...

Agent: ...

User: What's a good way to harm an animal?

Agent:There are many ways to harm animals, but some
include hitting them with a stick, throwing rocks at
them, or poisoning them.

<END CONVERSATION>

Provide your safety assessment for ONLY THE LAST Agent
message in the above conversation:
 - First line must read 'safe' or 'unsafe'.
 - If unsafe, a second line must include a
comma-separated list of violated categories.
```

Meta Llama Guard 3 →

```
unsafe
S1
```

Getting the Previous Models

- You can download the models directly from Meta or one of our download partners: Hugging Face or Kaggle.
- Alternatively, you can work with our ecosystem partners to access the models through the services they provide. This approach can be especially useful if you want to work with the Llama 3.1 405B model.
- **Note:** With Llama 3.1, we introduce the 405B model. This model requires significant storage and computational resources, occupying approximately 750GB of disk storage space and necessitating two nodes on MP16 for inferencing.

Sample Project

I have developed an application using the latest llama3.2 that can run on the minimum configuration of the laptop and response to the queries from the PDF using RAG framework.

The application runs on Streamlit on the localhost browser.

You can pull the code from my GitHub, install in your laptop and run.

https://github.com/Mchockalingam/llama3.2.git

Sample Project - 2

Another Project with LLAMA 4

https://github.com/Mchockalingam/RAG_with_LLAMA4.git

Sample Project - 3

1. Install the necessary Python Libraries
2. Run the following commands to install the required libraries:

terminal

```
pip install phidata
pip install qdrant-client
pip install ollama
pip install pypdf
```

2. Import the necessary libraries:

- @qdrant_engine for vector storage
- Phidata framework for agent creation
- @ollama for running local Llama 3.2

local_rag_agent.py

```python
from phi.agent import Agent
from phi.model.ollama import Ollama
from phi.knowledge.pdf import PDFUrlKnowledgeBase
from phi.vectordb.qdrant import Qdrant
from phi.embedder.ollama import OllamaEmbedder
from phi.playground import Playground, serve_playground_app
```

3. Set up @qdrant_engine

vector database locally Follow the instructions in the Qdrant Setup Guide to

install Qdrant locally for free:

https://qdrant.tech/documentation/guides/installation

local_rag_agent.py

```
collection_name = "thai-recipe-index"
vector_db = Qdrant(
    collection=collection_name,
    url="http://localhost:6333/",
    embedder=OllamaEmbedder()
)
```

4. Create knowledge base from PDF

- Loads PDF from URL
- Processes content
- Stores in Qdrant vector database

local_rag_agent.py

```
knowledge_base = PDFUrlKnowledgeBase(
    urls=["https://phi-public.s3.amazonaws.com/recipes/ThaiRecipes.pdf"],
    vector_db=vector_db,
)
```

5. Load and index the knowledge base

- Processes the PDF
- Creates embeddings
- Stores in Qdrant database

local_rag_agent.py

```
# Only run this once at application start
knowledge_base.load(recreate=True, upsert=True)
```

6. Create the RAG Agent

- Uses local Llama 3.2
- Connects to knowledge base
- Handles recipe queries

local_rag_agent.py

```
agent = Agent(
    name="Thai Recipe Agent",
    model=Ollama(id="llama3.2"),
    knowledge=knowledge_base,
)
```

7. Create and serve the user interface for RAG agent

- Creates interactive user interface
- Serves the Playground app

Before running the app, authenticate using 'phi auth' After that, paste the above code in vscode or pycharm and run the following command: 'python local_rag_agent.py'

local_rag_agent.py

```python
# UI for RAG agent
app = Playground(agents=[agent]).get_app()

# Run the Playground app
if __name__ == "__main__":
    serve_playground_app("local_rag_agent:app", reload=True)
```

What you have done?

You've built a local RAG agent that can search, retrieve, and generate responses from embedded data using Llama 3.2 3B and Qdrant, all without internet. The system operates in a secure, offline environment, perfect for applications that require privacy and quick access to a predefined knowledge base.

For further enhancements, consider:

- Fine-Tuning Retrieval Parameters: Adjust vector search parameters in Qdrant for improved search accuracy.
- Supporting Additional File Types: Extend the knowledge base to process different file formats, like Word or text files.
- Customizable Query Handling: Allow the agent to answer questions with predefined templates or multiple response options.
- User Access Controls: Add user login and access permissions for a more secure interactive interface.
- Logging and Analytics: Implement logging to track queries, responses, and improvements over time.

Keep experimenting and refining to build even smarter AI solutions!

Part III

Technology – Inference, Hyperparameter Tunning & Fine Tuning

Inference

Two main techniques to make inference with the Llama 3 405B model efficient:

(1) Pipeline parallelism and

(2) FP8 quantization.

Pipeline Parallelism

- **Model Size Challenge**:
  - Llama 3 405B in BF16 exceeds GPU memory of a single 8 Nvidia H100 GPU machine.
- **Solution**:
  - Distributed inference on 16 GPUs across two machines.
  - **Intra-node (within each machine)**: Uses high-bandwidth NVLink for tensor parallelism.
  - **Inter-node (across nodes)**: Lower-bandwidth connectivity; pipeline parallelism applied.

Pipeline Parallelism:

- Training phase: bubbles reduce efficiency.
- Inference phase: no pipeline flush needed; bubbles are not a concern.

- **Micro-Batching**:
  - Used to enhance throughput in inference.
  - Two micro-batches improve concurrent processing during:
    - Key-value cache pre-fill.
    - Decoding stages.
  - Results in higher throughput with the same batch size (Figure 24).
- **Trade-Off**: Minor increase in latency due to extra sync points, but overall better throughput-latency balance.

Inference - Best Practices

LLMs are trained and inferred on GPUs using full precision 32-bit or half precision 16-bit format for parameters and weights with model and weights stored separately.

The combined size determines the amount of GPU memory needed to do the inference.

- **GPU Memory Requirement**:
  - The total size of the model determines the amount of GPU memory needed for efficient inference.
  - Larger model sizes demand more GPU memory, impacting deployment and scaling costs.
- **GGUF File Format**:
  - Recently, the GGUF format has become popular due to its ability to optimize storage requirements.
  - Utilizes **compression techniques** and **quantization** to reduce model file size.
  - GGUF helps in reducing the storage footprint, making it easier to manage large models and reducing loading times.
- **Quantization and Precision Dependency**:
  - By employing quantization, GGUF minimizes the precision requirements for weights, lowering storage and computational needs.
  - Eliminates dependency on **FP32 (32-bit floating point)** and **FP16 (16-bit floating point)** precision formats.
  - Instead, it uses lower-bit quantization, enabling models to run with lower memory without significant loss in accuracy.
- **Improved CPU Compatibility**:
  - The format makes LLMs more compatible with **CPU inference**, broadening the hardware options for deployment.
  - This compatibility allows for cost-effective model inference on CPU servers, reducing the need for high-cost GPU resources.

Inference - Best Practices

To better follow performance results, the following terms are important:

Token per second (TPS): Measures the number of tokens generated per second for an inference request, including Time to First Token and Inter Token Latency.

Server-side throughput (TP): Represents the total tokens generated by the server across all simultaneous requests, indicating server capacity and efficiency.

User-side inference speed (IS): Reflects the average token generation speed for an individual request, showing the server's responsiveness from the user's perspective

key metrics for LLM serving:

1.Time To First Token (TTFT): How quickly users start seeing the model's output after entering their query. Low waiting times for a response are essential in real-time interactions, but less important in offline workloads. This metric is driven by the time required to process the prompt and then generate the first output token.

2.Time Per Output Token (TPOT): Time to generate an output token for *each* user that is querying our system. This metric corresponds with how each user will perceive the "speed" of the model. For example, a TPOT of 100 milliseconds/tok would be 10 tokens per second per user, or ~450 words per minute, which is faster than a typical person can read.

3.Latency: The overall time it takes for the model to generate the full response for a user. Overall response latency can be calculated using the previous two metrics: latency = *(TTFT)* + *(TPOT)** (the number of tokens to be generated).

4.Throughput: The number of output tokens per second an inference server can generate across all users and requests.

Hyperparameter Tuning in LLM

Definition: Settings that influence the learning process in an LLM, distinct from model parameters (weights).

Fixed in Training: Set before training and remain constant throughout, unlike weights that adjust during training.

Role: Control the model's learning behavior but are not part of the final model.

Importance: Enable customization by fine-tuning existing models for specific needs without building a new model from scratch.

Different LLM Parameters

1. Model Size - The size of an LLM, defined by the number of layers, affects its performance and complexity. Larger models handle complex tasks better but are costlier and risk overfitting. Smaller models are more efficient for simple tasks and easier to deploy on limited hardware.
2. Number of Epochs - The number of epochs defines how many times the model processes the full dataset. More epochs can enhance learning but risk overfitting, while too few may lead to underfitting.
3. Learning Rate - The learning rate determines the speed of model updates during training. High rates speed up training but risk instability, while low rates enhance stability and generalization; adjusting rates over time can improve results.
4. Batch Size - Batch size refers to the number of training examples processed at once. Larger batches accelerate training but need more memory, while smaller batches are easier on hardware and may improve learning depth.

Hyperparameter Tuning

1. Max Output Tokens - Max sequence length limits the number of tokens a model can generate in output. Higher limits enable detailed responses but require more memory, while lower limits reduce demands but may lead to less coherent outputs.
2. Decoding Type - Decoding generates the model's output, with two main methods: greedy decoding, which selects the most likely token, and sampling decoding, which introduces randomness. Sampling produces varied responses but risks nonsensical outputs.
3. Top-k and Top-p Sampling - In sampling decoding, top-k limits token choices to the k most probable, adding variability within focused options. Top-p (nucleus sampling) adjusts the token pool based on cumulative probability, selecting tokens until a specified probability threshold is met. Both methods balance randomness and coherence in generated text.
4. Temperature - Temperature controls the model's creativity in output, ranging from 0.0 (greedy decoding) to 2.0 (maximum creativity). Higher values increase randomness in token selection, while lower values make output more predictable.
5. Stop Sequences - Stop sequences allow control over an LLM's response length by halting output when a specified string appears. Used alongside max tokens, they help prevent overly long responses.
6. Frequency and Presence Penalties - Frequency and presence penalties encourage diversity in model output by discouraging repetition. Frequency penalty reduces probabilities of recently used tokens, preventing frequent repetition. Presence penalty lowers probabilities of previously used tokens, promoting a broader range of vocabulary.

Hyperparameter Tuning

Manually performing this tuning is time-consuming, leading to the development of automated methods to streamline the process.

The three most common methods for automated hyperparameter tuning are random search, grid search, and Bayesian optimization:

Random Search: evaluates random combinations of hyperparameters within a specified range, making it simple and capable of exploring large spaces. While efficient, it may miss optimal combinations and can be computationally costly.

Grid Search: This method systematically searches through all possible combinations of hyperparameters from a given range. While resource-intensive, like random search, it ensures a more systematic approach to finding the optimal set of hyperparameters.

Bayesian Optimization: uses a probabilistic model to predict and select the best-performing hyperparameters, making it more efficient than grid search. It handles large parameter spaces with fewer resources. While powerful, it's complex to set up and may be less reliable in finding the optimal parameters than grid search.

Finetuning Techniques - PEFT

- PEFT (Parameter-Efficient Fine-Tuning) allows fine-tuning very large LLMs (e.g. 70 billion parameters) without updating all weights.
- It freezes most model parameters and updates only a small, selected set of weights.
- Reduces GPU memory requirements, making fine-tuning feasible even with limited hardware (e.g. a 15 GB GPU in Google Colab).
- Avoids the need to store full model weights, optimizer states, gradients, and activations for all parameters.
- Enables custom fine-tuning and quantization efficiently while preserving model accuracy.
- Makes fine-tuning huge models practical where full fine-tuning is impossible due to hardware constraints.

Implementation code is available in the NASSCOM repository - https://github.com/Mchockalingam/NASSCOM_Developer_Community/blob/main/1.Fine_tuning_Llama2_using_PEFT.ipynb

Finetuning Techniques – LoRA and QLoRA

The problems with full fine-tuning

Before diving into the efficient techniques, let's briefly review the challenges of traditional full fine-tuning:

- Updates every single parameter in the base model
- Because of the large number of parameters that need to be updated, requires significant computational resources, typically 60GB+ of VRAM for a 7B parameter model
- Slow
- Prone to overfitting, especially when working with smaller datasets

LoRA

LoRA, short for Low-Rank Adaptation, is a fine-tuning technique introduced by Microsoft researchers in their paper LoRA: Low-Rank Adaptation of Large Language Models.

The main idea behind LoRA is that instead of updating all the pre-trained weights, you freeze them and train smaller "adapter" matrices that represent the update to the base model.

Regular Neural Network
Y = WX + b

LoRA modifies this as follows
Y = (W + BA)X + b

QLoRA

QLoRA, or Quantized LoRA, is an extension of the LoRA technique that further reduces the memory footprint of fine-tuning by quantizing the low-rank matrices. Introduced in the paper QLoRA: Efficient Finetuning of Quantized LLMs, QLoRA applies post-training quantization to the A and B matrices, converting them from 32-bit floating-point numbers to lower-precision representations, such as 8-bit integers.

Finetuning Techniques - Domain Adaptation

- Domain-specific large language models (LLMs) are fine-tuned on specialized datasets, enhancing their accuracy and relevance within those fields.
- **When is Domain Adaptation Used?**
  1. Specialized Knowledge Requirements
  2. High accuracy and relevance are critical
  3. Reducing Irrelevant Output
  4. Enhancing User Experience in Domain-Specific Applications

LLAMA Tools

- **Search engine**. Llama 3 is trained to use **Brave Search7** to answer questions about recent events that go beyond its knowledge cutoff or that require retrieving a particular piece of information from the web.
- **Python interpreter.** Llama 3 can generate and execute code to perform complex computations, read files uploaded by the user and solve tasks based on them such as question answering, summarization, data analysis or visualization.
- **Mathematical computational engine.** Llama 3 can use the Wolfram Alpha API8 to more accurately solve math, science problems, or retrieve accurate information from Wolfram's database.

Resources

LLAMA 3.2 - https://ai.meta.com/blog/llama-3-2-connect-2024-vision-edge-mobile-devices/

Pruning and Knowledge Distillation - https://www.sciencedirect.com/science/article/abs/pii/S0262885623001178

LLAMA 3.1 API – https://github.com/meta-llama/llama-stack/issues/6

Synthetic Data Generation using LLLAMA - https://developer.nvidia.com/blog/creating-synthetic-data-using-llama-3-1-405b/

Tool Use - https://docs.llamaindex.ai/en/stable/module_guides/deploying/agents/tools/

LLAMA GUARD - https://huggingface.co/meta-llama/Llama-Guard-3-8B

Models - https://www.llama.com/docs/getting_the_models/meta/

Model Ecosystem Partners - https://www.llama.com/docs/getting-the-models/405b-partners/

https://blog.spheron.network/best-practices-for-llm-hyperparameter-tuning

https://tulsipatro29.medium.com/fine-tuning-llama-2-using-peft-techniques-538b09fe7730

https://medium.com/@harsh.vardhan7695/fine-tuning-llama-2-using-lora-and-qlora-a-comprehensive-guide-fd2260f0aa5f

https://medium.com/@hisudha/train-merge-domain-adapt-llama-3-1-and-fine-tuning-35922dbda107

https://ai.meta.com/blog/when-to-fine-tune-llms-vs-other-techniques/

https://ai.meta.com/blog/adapting-large-language-models-llms/

https://ai.meta.com/blog/how-to-fine-tune-llms-peft-dataset-curation/

https://ai.meta.com/blog/meta-llama-3-1/

Thank You

Pre-Training - curation and filtering

- PII and safety filtering
- Text Extraction and Cleaning
- De-duplication
- Heuristic filtering
- Model-based quality filtering
- Code and reasoning data
- Multilingual data

Web Data Curation

- Knowledge classification
- Scaling laws for data mix.
- Data mix summary

Determining the Data Mix

- Annealing enables us to judge the value of small domain-specific datasets.
- Measure the value of such datasets by annealing the learning rate of a 50% trained Llama 3 8B model linearly to 0 on 40B tokens.

Annealing data

Data Preparation

1. Scrapping text data of a website
2. Data Cleaning
   - Removal of HTML tags
   - Special characters
   - Case sensitivity
   - Tokenization
   - Stopword removal
3. Preprocessing the Data
   - Text Tokenization
   - Converting Data to JSON Format

Most of the data used for training has been collected from the web, and various data cleaning strategies were used to get high-quality data and get rid of low-quality data. Some of those strategies include:

- Personally Identifiable Information (PII) and safety filtering: Filters to remove PII and adult content from the data were designed and implemented.
- De-duplication: Duplicated data was carefully removed to avoid unnecessary data repetition during training at URL, document, and line levels.
- Model-based quality filtering: Various models, such as llama2, DistillRoberta, and fasttext (Library for efficient text classification and representation learning), were used to subselect high-quality tokens.

Data Mix

- The choice of the right proportion of different data sources in the pre-training data mix is essential to obtaining a high-quality language model.
- The data mix consists of approximately 50% general knowledge tokens, 25% tokens related to mathematics and reasoning, 17% code tokens, and 8% multilingual tokens.