# LLM / GenAI Bootstrap

## Local and Managed LLMs with pure Java

airhacks.live

"It's not work if you like it"
…so I never worked.  #java

# airhacks.fm

## #296 Exploring ONNX, Embedding Models, and Retrieval Augmented Generation (RAG) with Langchain4j

[episode link]

An airhacks.fm

Dmytro pre
for running
handling, e
longer text
generation
and data so
powered ap
embedding
embedding
outputs usi
specialized

Dmytro Liubars

## #313 Revolutionizing AI with Java: From LLMs to Vector APIs

[episode link] Listen on Ap

An airhacks.fm c

Alfonso previ
projects for r
multiplication
qwen or gem
for Java Com
integrating w
optimizations
but limited w
improvement
in enterprise

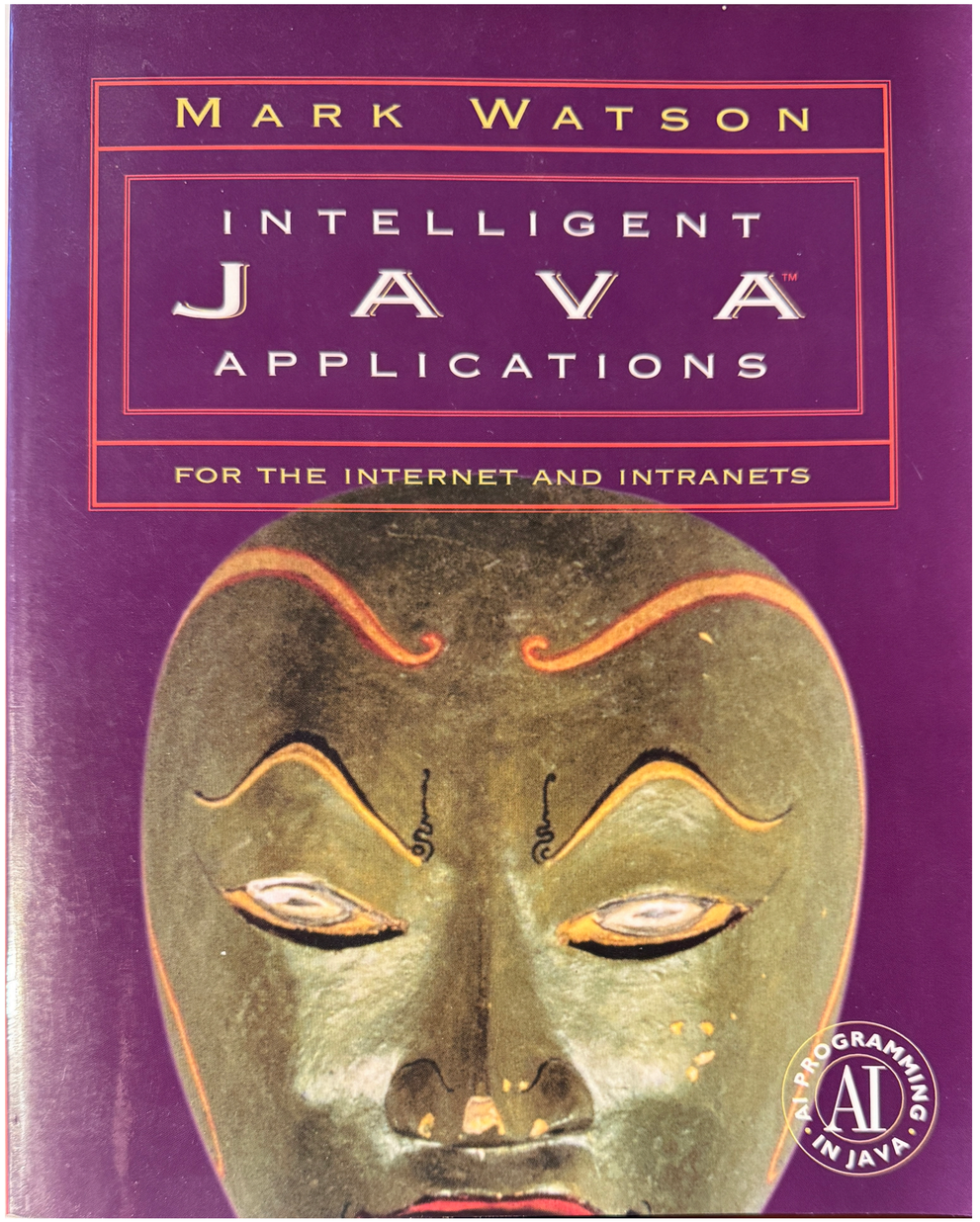
## #315 The AI Revolution in Java Development and Devoxx Genie

[episode link] Listen on Apple Podcasts LISTEN ON Spotify Listen on Google Podcasts [RSS]

An airhacks.fm conversation with Stephan Janssen (@Stephan007) about:

Stephan previously appeared on "#254 How JavaPolis and Devoxx Happened", discussion on the AI rev
development of an AI-assisted photo sharing application, creation of the Devoxx Genie IntelliJ plugin f
advantages of Claude 3.5 from Anthropic, use of local AI models in development environments, integra
langchain4j and its adoption by Red Hat, development of Java-based AI tools like Lama3.java, jlama an
AI models in software development, challenges and opportunities for junior and senior developers in AI
importance of understanding cloud services and cost structures when using AI, potential future of pro
generation, discussion on maintaining and improving AI-generated code, exciting developments in Jav
and tornadovm, potential for running AI models directly on Java without external dependencies, consi
and integration, the need for promoting Java's capabilities in AI development, potential for Visual Stud

airhacks.live


MARK WATSON
INTELLIGENT
JAVA™
APPLICATIONS
FOR THE INTERNET AND INTRANETS
AI PROGRAMMING IN JAVA
AI

(1997)

airhacks.TV

with the time machine, “100 episodes ago segment”

…any questions left?

youtube.com/
@bienadam/shorts


airhacks.live

NEW https://discord.gg/airhacks

# java …for LLMs?

| Total | | | | | |
|---|---|---|---|---|---|
| | Energy | | Time | | Mb |
| (c) C | 1.00 | (c) C | 1.00 | (c) Pascal | 1.00 |
| (c) Rust | 1.03 | (c) Rust | 1.04 | (c) Go | 1.05 |
| (c) C++ | 1.34 | (c) C++ | 1.56 | (c) C | 1.17 |
| (c) Ada | 1.70 | (c) Ada | 1.85 | (c) Fortran | 1.24 |
| (v) Java | 1.98 | (v) Java | 1.89 | (c) C++ | 1.34 |
| (c) Pascal | 2.14 | (c) Chapel | 2.14 | (c) Ada | 1.47 |
| (c) Chapel | 2.18 | (c) Go | 2.83 | (c) Rust | 1.54 |
| (v) Lisp | 2.27 | (c) Pascal | 3.02 | (v) Lisp | 1.92 |
| (c) Ocaml | 2.40 | (c) Ocaml | 3.09 | (c) Haskell | 2.45 |
| (c) Fortran | 2.52 | (v) C# | 3.14 | (i) PHP | 2.57 |
| (c) Swift | 2.79 | (v) Lisp | 3.40 | (c) Swift | 2.71 |
| (c) Haskell | 3.10 | (c) Haskell | 3.55 | (i) Python | 2.80 |
| (v) C# | 3.14 | (c) Swift | 4.20 | (c) Ocaml | 2.82 |
| (c) Go | 3.23 | (c) Fortran | 4.20 | (v) C# | 2.85 |
| (i) Dart | 3.83 | (v) F# | 6.30 | (i) Hack | 3.34 |
| (v) F# | 4.13 | (i) JavaScript | 6.52 | (v) Racket | 3.52 |
| (i) JavaScript | 4.45 | (i) Dart | 6.67 | (i) Ruby | 3.97 |
| (v) Racket | 7.91 | (v) Racket | 11.27 | (c) Chapel | 4.00 |
| (i) TypeScript | 21.50 | (i) Hack | 26.99 | (v) F# | 4.25 |
| (i) Hack | 24.02 | (i) PHP | 27.64 | (i) JavaScript | 4.59 |
| (i) PHP | 29.30 | (v) Erlang | 36.71 | (i) TypeScript | 4.69 |
| (v) Erlang | 42.23 | (i) Jruby | 43.44 | (v) Java | 6.01 |
| (i) Lua | 45.98 | (i) TypeScript | 46.20 | (i) Perl | 6.62 |
| (i) Jruby | 46.54 | (i) Ruby | 59.34 | (i) Lua | 6.72 |
| (i) Ruby | 69.91 | (i) Perl | 65.79 | (v) Erlang | 7.20 |
| (i) Python | 75.88 | (i) Python | 71.90 | (i) Dart | 8.64 |
| (i) Perl | 79.58 | (i) Lua | 82.91 | (i) Jruby | 19.84 |

GraalVM™

reduces RAM footprint

https://sites.google.com/view/energy-efficiency-languages/results?authuser=0

## Multithreading Support

One of the advantage of Deep Java Library (DJL) is Multi-threaded inference support. It can help to increase the throughput of your inference on multi-core CPUs and GPUs and reduce memory consumption compared to Python.

http://djl.ai/docs/development/inference_performance_optimization.html

### Performance

Before DJL, running predictions with this model and such high-dimensional data used to take around 24 hours and had multiple memory issues. DJL reduced the prediction time on this model by **85%**, from around one day to a couple of hours. DJL worked out of the box without spending any time on engineering tasks, such as memory tuning. In contrast, prior to DJL, we spent more than two weeks in memory tuning.

## More about DJL

Deep Java Library (DJL) is an open source library to build and deploy deep learning in Java. This project launched in December 2019 and is widely used among teams at Amazon. This effort was inspired by other DL frameworks, but was developed from the ground up to better suit Java development practices. DJL is framework agnostic, with support for Apache MXNet, PyTorch, TensorFlow 2.x (experimental), and fastText (experimental). Additionally, DJL offers a repository of pre-trained models in our ModelZoo that simplifies implementation and streamlines model sharing across projects.

https://aws.amazon.com/blogs/opensource/how-amazon-retail-systems-run-machine-learning-predictions-with-apache-spark-using-deep-java-library/

Total

| | Energy |
|---|---|
| (c) C | 1.00 |
| (c) Rust | 1.03 |
| (c) C++ | 1.34 |
| (c) Ada | 1.70 |
| (v) Java | 1.98 |
| (c) Pascal | 2.14 |
| (c) Chapel | 2.18 |
| (v) Lisp | 2.27 |
| (c) Ocaml | 2.4 |
| (c) Fortran | 2 |
| (c) Swift | 9 |
| (c) Haskell | 0 |
| (v) C# | |
| (c) | 3.1 |
| (i) | 3.8 |
| F# | 4.13 |
| | 4.45 |
| | 7.91 |
| | 21.50 |
| | 24.02 |
| (i) PHP | 29.30 |
| (v) Erlang | 42.23 |
| (i) Lua | 45.98 |
| (i) Jruby | 46.54 |
| (i) Ruby | 6 |
| (i) Python | .88 |
| (i) Perl | 9.5 |

| | T |
|---|---|
| (c) C | 1. |
| (c) Rust | 1.0 |
| (c) C++ | 1.56 |
| (c) Ada | 1.5 |
| Java | |
| (c) Chapel | 2. |
| (c) | 2. |
| (c) | 2 |
| (c) Ocaml | 3.09 |
| C# | 3.14 |
| | 3.40 |
| Haskell | 3.55 |
| (c) Swift | 4.20 |
| (c) Fortran | 4.20 |
| (v) F# | 6.30 |
| (i) JavaScript | 6.52 |
| (i) Dart | 6.67 |
| (v) Racket | 1.27 |
| (i) Hack | |
| (i) PHP | 27 |
| (v) Erlang | 36.71 |
| (i) Jruby | 43.44 |
| | 6.20 |
| Ruby | 9.34 |
| Perl | 65.79 |
| Python | 71.90 |
| (i) | 82.91 |

| | Mb |
|---|---|
| Pascal | 1.00 |
| | 1.05 |
| (c) | 1.17 |
| (c) an | 1.24 |
| C++ | 1.34 |
| (c) Ada | 1.47 |
| (c) Rust | 1.54 |
| (v) Lisp | 1.92 |
| (c) Haskell | 2.45 |
| (i) PHP | 2.57 |
| (c) Swift | |
| (i) Python | 2.80 |
| (c) Ocaml | 2.82 |
| (v) | 85 |
| Hack | 4 |
| Racket | |
| (i) Ruby | 3.97 |
| (c) pel | 4.00 |
| (v) | 4.25 |
| (i) JavaScript | 4.59 |
| (i) TypeScript | 4.69 |
| (v) Java | 6.01 |
| (i) Perl | 6.62 |
| (i) Lua | 6.72 |
| (v) Erlang | 7.20 |
| (i) Dart | 8.64 |
| (i) Jruby | 19.84 |

partially relevant

https://sites.g e.com/v rgy-efficiency-languages/results?authuser=0

GraalVM™

reduces RAM footprint

# serverless facts

The calculations below exclude free tier discounts.

Architecture

Arm

Number of requests

2

Unit

per second

Duration of each request (in ms)

Duration is calculated from the time your code begins executing until it returns or otherwise terminates.

100

Amount of memory allocated

Enter the amount between 128 MB and 10 GB

Value

2

Unit

GB

Amount of ephemeral storage allocated

Enter the amount between 512 MB and 10,240 MB. The first 512 MB are at no additional charge, you only pay for any additional storage that you configure for the function.

Value

512

Unit

MB

▶ Show calculations

**Total Upfront cost:** 0.00 USD
**Total Monthly cost:** 15.07 USD

Show Details ▼

Cancel

Save and view summary

Save and add servi

The calculations below exclude free tier discounts.

Architecture

Arm

Number of requests

2

Unit

per second

Duration of each request (in ms)

Duration is calculated from the time your code begins executing until it returns or otherwise terminates.

10000

Amount of memory allocated

Enter the amount between 128 MB and 10 GB

Value

2

Unit

GB

Amount of ephemeral storage allocated

Enter the amount between 512 MB and 10,240 MB. The first 512 MB are at no additional charge, you only pay for any additional storage that you configure for the function.

Value

512

Unit

MB

▶ Show calculations

**Total Upfront cost:** 0.00 USD
**Total Monthly cost:** 1,402.66 USD

Show Details

Cancel

Save and view summary

Save and add servi

dam-bien

The calculations below exclude free tier discounts.

Architecture

Arm

Number of requests

2

Unit

per second

Duration of each request (in ms)

Duration is calculated from the time your code begins executing until it returns or otherwise terminates.

10000

Amount of memory allocated

Enter the amount between 128 MB and 10 GB

Value

128

Unit

MB

Amount of ephemeral storage allocated

Enter the amount between 512 MB and 10,240 MB. The first 512 MB are at no additional charge, you only pay for any additional storage that you configure for the function.

Value

512

Unit

MB

▶ Show calculations

**Total Upfront cost:** 0.00 USD
**Total Monthly cost:** 88.65 USD

Show Details

Cancel

Save and view summary

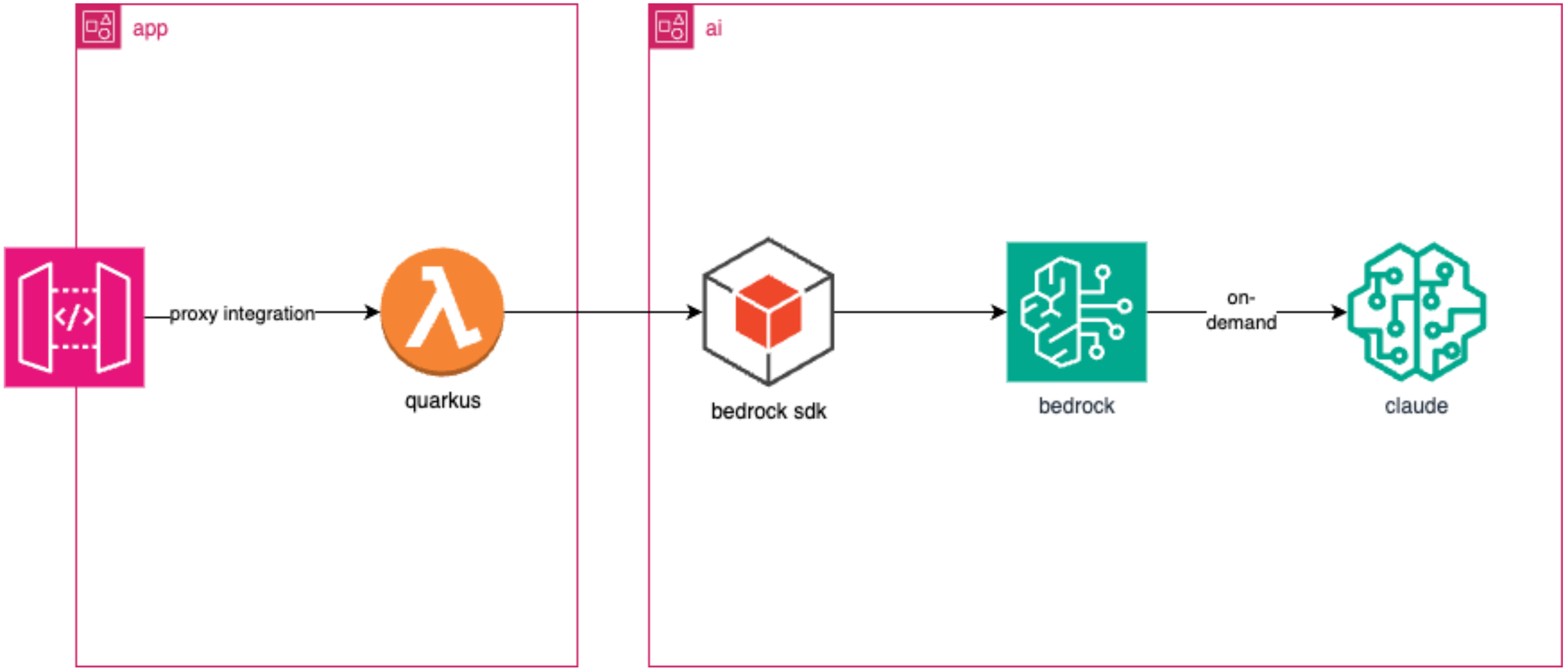
Save and add servi

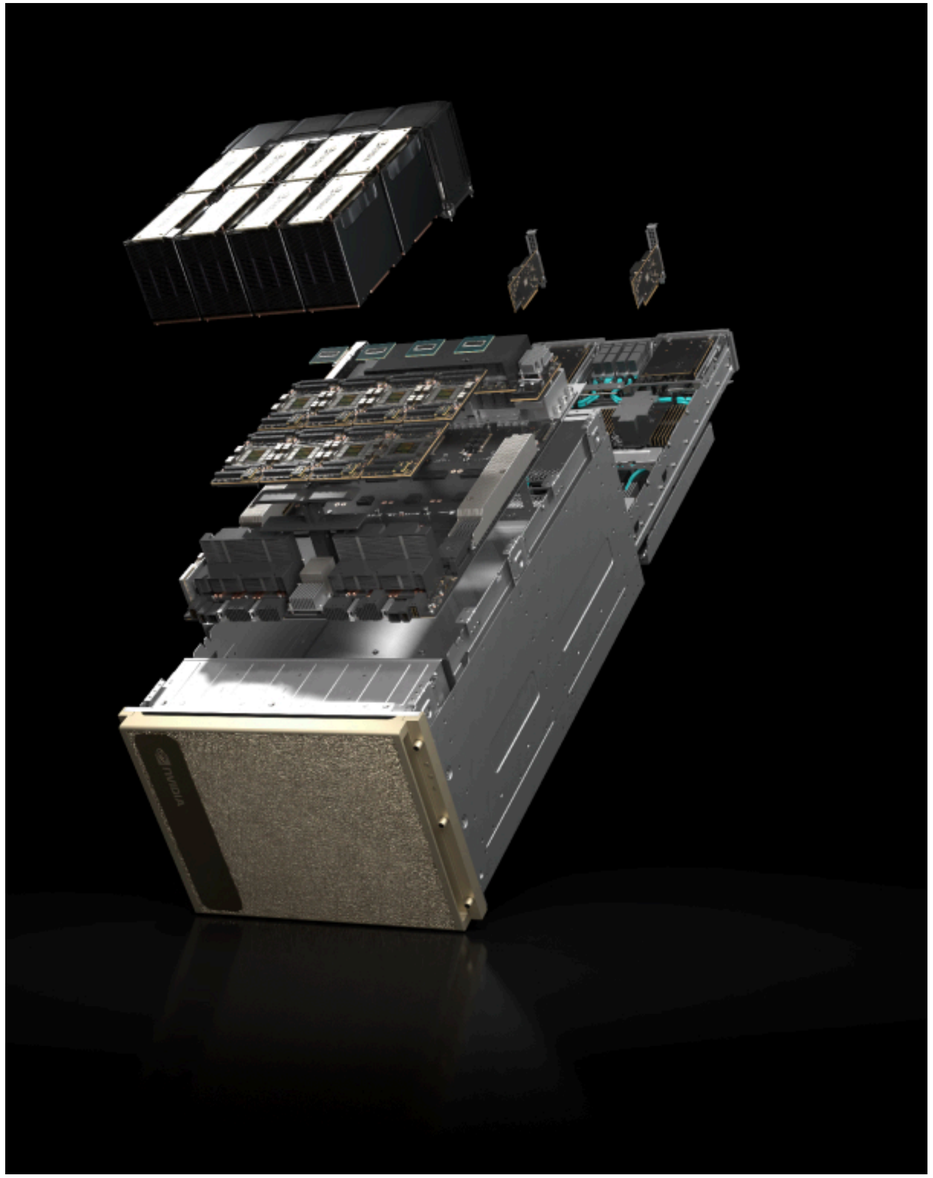
am-bien

# Amazon Bedrock

- fully managed service
- serverless
- provides access to FMs via a REST API
- integrated with IAM

# Amazon Bedrock



https://www.nvidia.com/en-us/data-center/dgx-h200/

# LLMs

# NDArray

## The ndarray data structure [ edit ]

The core functionality of NumPy is its "ndarray", for *n*-dimensional array, data structure. These arrays are strided views on memory.[9] In contrast to Python's built-in list data structure, these arrays are homogeneously typed: all elements of a single array must be of the same type.

Such arrays can also be views into memory buffers allocated by C/C++, Python, and Fortran extensions to the CPython interpreter without the need to copy data around, giving a degree of compatibility with existing numerical libraries. This functionality is exploited by the SciPy package, which wraps a number of such libraries (notably BLAS and LAPACK). NumPy has built-in support for memory-mapped ndarrays.[9]

https://en.wikipedia.org/wiki/NumPy

# LLMs

- training vs. inference
- lossy compression of the internet
- prediction of the next word in sequence
- pre-trained models vs. e.g. instruct models

# Temperature

“how the model selects its next words during text generation”

controls the randomness and creativity in a language model's output:

- 0.0: Deterministic (always picks highest probability)
- 0.7: Balanced (common for chat)
- 1.0: Standard scaling
- 2.0: Very random/creative

# Top-K

**Top-*p* sampling**, also called **nucleus sampling**, is a technique for autoregressive language model decoding proposed by Ari Holtzman in 2019.[1] Before the introduction of nucleus sampling, maximum likelihood decoding and beam search were the standard techniques for text generation, but, both of these decoding strategies are prone to generating texts that are repetitive and otherwise unnatural.[2] Top-*p* sampling avoids this by setting a threshold $\mathrm{p}$ and then restricting the sampling to the set of most probable tokens with cumulative probability more than $\mathrm{p}$. Then, probabilities of the token from this set are rescaled to sum up to 1, the rest of tokens are rejected.

Top-*k* sampling is similar except that the sample is taken from the *k*-highest probability tokens regardless of their cumulative probability. The advantage of top-*p* sampling is that one avoids the difficult problem of choosing the optimal value of $\mathrm{k}$ which can vary depending on the shape of the output distribution and the particular task and dataset.[3]

The top-*p* sampling technique is used in popular large language model applications like ChatGPT and is implemented in language modeling frameworks like Hugging Face and Cohere.[4]

https://en.wikipedia.org/wiki/Top-p_sampling

# Top-K

- Simpler to implement
- Predictable performance
- Might cut off relevant tokens in flat distributions
- Less adaptable to different contexts

# Top-K

- Common range: 40 to 50
- Conservative/focused: 10 to 20
- Balanced: 20 to 40
- Creative/diverse: 50 to 100

# Top-P

- Adapts to probability distribution
- More natural language variation
- Variable computation needs
- Slightly more complex to implement

# Top-P

- Common range: 0.9 to 0.95
- Conservative/focused: 0.1 to 0.3
- Balanced: 0.7 to 0.9
- Creative/diverse: 0.95 to 1.0

# Top-P + Top-K

- Top-K: hard limit
- Top-P: fine grained filtering

# “Enterprise Configuration”

**Temperature: 0.1 - 0.3**

Lower temperature makes the model stick to more likely/confident predictions

This reduces creativity but increases factual accuracy

**Top-P (nucleus sampling): 0.1 - 0.3**

Lower values restrict the model to only the most probable tokens

This makes outputs more focused and deterministic

**Top-K: 10 - 20**

A smaller K limits the pool of possible tokens

This helps prevent the model from selecting unlikely tokens

# Quantization

Quantization is a technique to reduce the computational and memory costs of running inference by representing the weights and activations with low-precision data types like 8-bit integer (int8) instead of the usual 32-bit floating point (float32).

Reducing the number of bits means the resulting model requires less memory storage, consumes less energy (in theory), and operations like matrix multiplication can be performed much faster with integer arithmetic. It also allows to run models on embedded devices, which sometimes only support integer data types.

https://huggingface.co/docs/optimum/en/concept_guides/quantization

# Quantization

The basic idea behind quantization is quite easy: going from high-precision representation (usually the regular 32-bit floating-point) for weights and activations to a lower precision data type. The most common lower precision data types are:

float16, accumulation data type float16

bfloat16, accumulation data type float32

int16, accumulation data type int32

int8, accumulation data type int32

https://huggingface.co/docs/optimum/en/concept_guides/quantization

# Java 20

```
short halved = Float.floatToFloat16(1.5f);
```

https://huggingface.co/docs/optimum/en/concept_guides/quantization

# Quantization

- model memory footprint reduction
- reducing the precision of the weights
- 8-bit or 4-bit, even 1-bit instead of the typical 32-bit floating point
- Java Vector API may introduce float16

# Base Model

- trained on broad data
- prediction of the next word in sequence
- base model vs. fine tuning
- general vs. specific

# Fine-tuned model

- additional training on specific data
- less but high quality data
- chats: data labelling by humans (high quality data)
- fine-tuning adapts a base model's general knowledge to specific applications or domains

# Model Distillation

In machine learning, **knowledge distillation** or **model distillation** is the process of transferring knowledge from a large model to a smaller one. While large models (such as very deep neural networks or ensembles of many models) have more knowledge capacity than small models, this capacity might not be fully utilized. It can be just as computationally expensive to evaluate a model even if it utilizes little of its knowledge capacity. Knowledge distillation transfers knowledge from a large model to a smaller one without loss of validity. As smaller models are less expensive to evaluate, they can be deployed on less powerful hardware (such as a mobile device).[1]

Model distillation is not to be confused with model compression, which describes methods to decrease the size of a large model itself, without training a new model. Model compression generally preserves the architecture and the nominal parameter count of the model, while decreasing the bits-per-parameter.

Knowledge distillation has been successfully used in several applications of machine learning such as object detection,[2] acoustic models,[3] and natural language processing.[4] Recently, it has also been introduced to graph neural networks applicable to non-grid data.[5]

# GenAI

# GenAI

**Generative artificial intelligence** (**generative AI**, **GenAI**,[1] or **GAI**) is artificial intelligence capable of generating text, images, videos, or other data using generative models,[2] often in response to prompts.[3][4] Generative AI models learn the patterns and structure of their input training data and then generate new data that has similar characteristics.[5][6]

Improvements in transformer-based deep neural networks, particularly large language models (LLMs), enabled an AI boom of generative AI systems in the early 2020s. These include chatbots such as ChatGPT, Copilot, Gemini and LLaMA, text-to-image artificial intelligence image generation systems such as Stable Diffusion, Midjourney and DALL-E, and text-to-video AI generators such as Sora.[7][8][9][10] Companies such as OpenAI, Anthropic, Microsoft, Google, and Baidu as well as numerous smaller firms have developed generative AI models.[3][11][12]

https://en.wikipedia.org/wiki/Generative_artificial_intelligence

# Transformer

A **transformer** is a deep learning architecture developed by Google and based on the multi-head attention mechanism, proposed in a 2017 paper "Attention Is All You Need".[1] Text is converted to numerical representations called tokens, and each token is converted into a vector via looking up from a word embedding table.[1] At each layer, each token is then contextualized within the scope of the context window with other (unmasked) tokens via a parallel multi-head attention mechanism allowing the signal for key tokens to be amplified and less important tokens to be diminished. The transformer paper, published in 2017, is based on the softmax-based attention mechanism proposed by Bahdanau *et. al.* in 2014 for machine translation,[2][3] and the Fast Weight Controller, similar to a transformer, proposed in 1992.[4][5][6]

https://en.wikipedia.org/wiki/Transformer_(deep_learning_architecture)

# 2017: ”Attention Is All You Need"

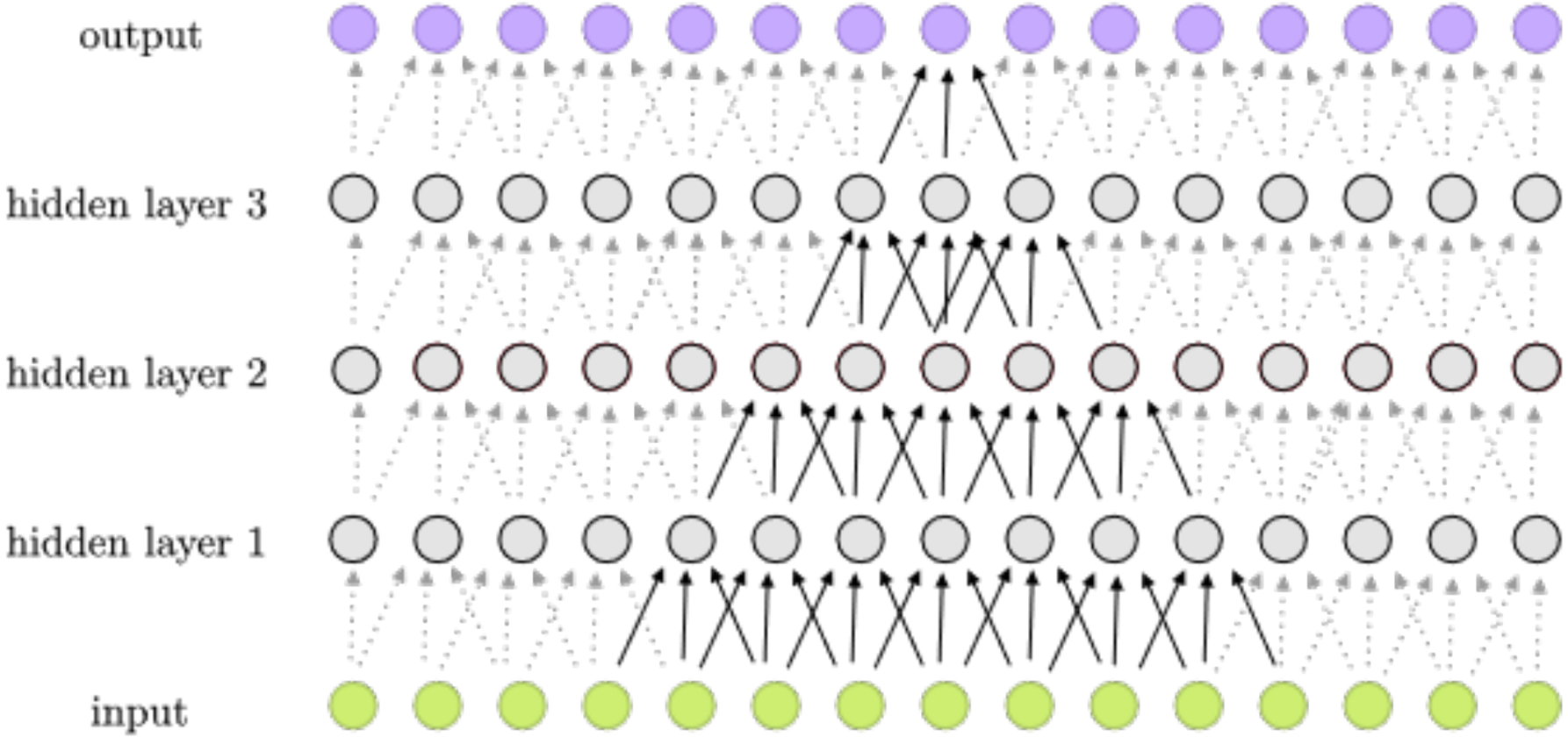
- no recurrence or convolutions
- dependencies between input and output
- attention operation: query and set of key-value pairs to an output
- multi-head attention: parallel attention

# Convolution

- part of the Convolutional Neural Networks (CNNs)
- deep learning / image recognition
- “sliding” with one filter over the entire input
- slides a filter over data, multiplying and summing
- pattern detection / image recognition
- similar to reduce()

https://en.wikipedia.org/wiki/Convolutional_neural_network

# Convolution



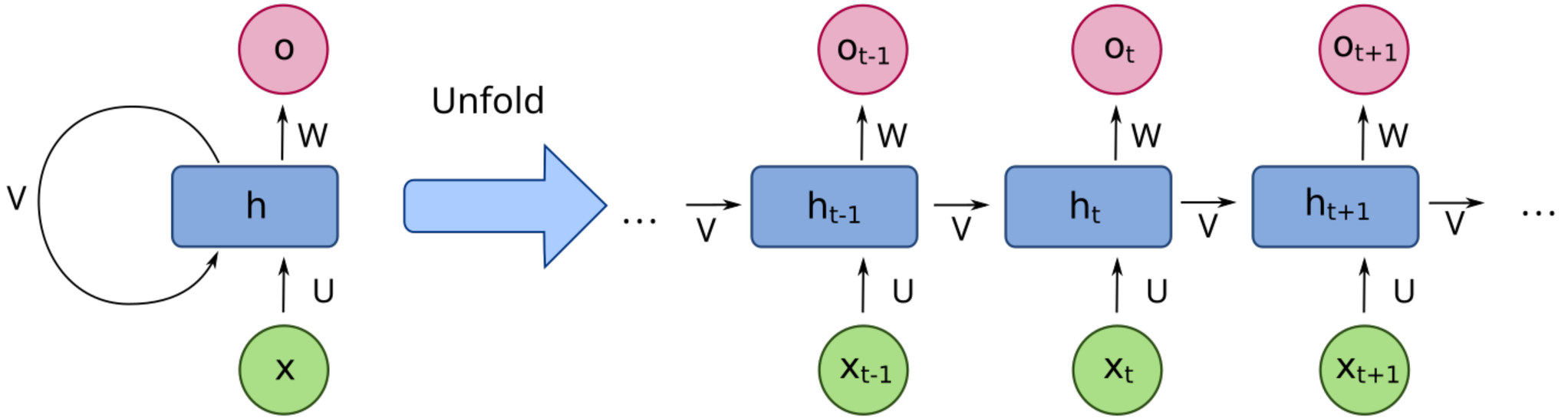

https://en.wikipedia.org/wiki/Convolutional_neural_network

# Recurrence

- a loop
- a step uses input from previous steps
- the memory of neural networks
- context or history
- processing a word considering previous words
- Recurrent Neural Network (RNN)

https://en.wikipedia.org/wiki/Recurrent_neural_network

# Recurrence (RNN)




From bottom to top: input state, hidden state, output state. U, V, W are the weights of the network. Compressed diagram on the left and the unfold version of it on the right.

https://en.wikipedia.org/wiki/Recurrent_neural_network

# Recurrence vs. Transformers

- Recurrence:
  - sequential
  - output is fed to input
  - a loop
  - memory efficient
- Transformers:
  - parallel
  - global connection

# Token

- AWS Bedrock: Use 6 characters per token as an approximation for the number of tokens.
- https://platform.openai.com/tokenizer

# softmax function

- converts numbers into probabilities (from 0 to 1)
- focus on highest values
- output sum to 1 (e.g. 0.8 + 0.2)
- arbitrary input values are converted into probability distribution

https://en.wikipedia.org/wiki/Softmax_function

# Vector

- floating-point numbers representing different aspects or features of the word/text
- each number corresponds to a specific dimension of meaning
- a vector captures the semantics
- is created with “embedding”
- an embedding model and a chat model have to be compatible

# Vector Database

# Vector Database

A **vector database**, **vector store** or **vector search engine** is a database that can store vectors (fixed-length lists of numbers) along with other data items. Vector databases typically implement one or more Approximate Nearest Neighbor (ANN) algorithms,[1][2] so that one can search the database with a query vector to retrieve the closest matching database records.

Vectors are mathematical representations of data in a high-dimensional space. In this space, each dimension corresponds to a feature of the data, with the number of dimensions ranging from a few hundred to tens of thousands, depending on the complexity of the data being represented. A vector's position in this space represents its characteristics. Words, phrases, or entire documents, as well as images, audio, and other types of data, can all be vectorized.[3]

These feature vectors may be computed from the raw data using machine learning methods such as feature extraction algorithms, word embeddings[4] or deep learning networks. The goal is that semantically similar data items receive feature vectors close to each other.

Vector databases can be used for similarity search, multi-modal search, recommendations engines, large language models (LLMs), etc.[5]

https://en.wikipedia.org/wiki/Vector_database

# Vector Database (contd.)

Vector databases are also often used to implement Retrieval-Augmented Generation (RAG), a method to improve domain-specific responses of large language models. The retrieval component of a RAG can be any search system, but is most often implemented as a vector database. Text documents describing the domain of interest are collected, and for each document or document section, a feature vector (known as an "embedding") is computed, typically using a deep learning network, and stored in a vector database. Given a user prompt, the feature vector of the prompt is computed, and the database is queried to retrieve the most relevant documents. These are then automatically added into the context window of the large language model, and the large language model proceeds to create a response to the prompt given this context.[6]

https://en.wikipedia.org/wiki/Prompt_engineering

# RAG

# RAG

**Retrieval-augmented generation** [ edit ]

Retrieval-Augmented Generation (RAG) is a two-phase process involving document retrieval and answer formulation by a Large Language Model (LLM). The initial phase utilizes dense embeddings to retrieve documents. This retrieval can be based on a variety of database formats depending on the use case, such as a vector database, summary index, tree index, or keyword table index.[51]

In response to a query, a document retriever selects the most relevant documents. This relevance is typically determined by first encoding both the query and the documents into vectors, then identifying documents whose vectors are closest in Euclidean distance to the query vector. Following document retrieval, the LLM generates an output that incorporates information from both the query and the retrieved documents.[52] This method is particularly beneficial for handling proprietary or dynamic information that was not included in the initial training or fine-tuning phases of the model. RAG is also notable for its use of "few-shot" learning, where the model uses a small number of examples, often automatically retrieved from a database, to inform its outputs.

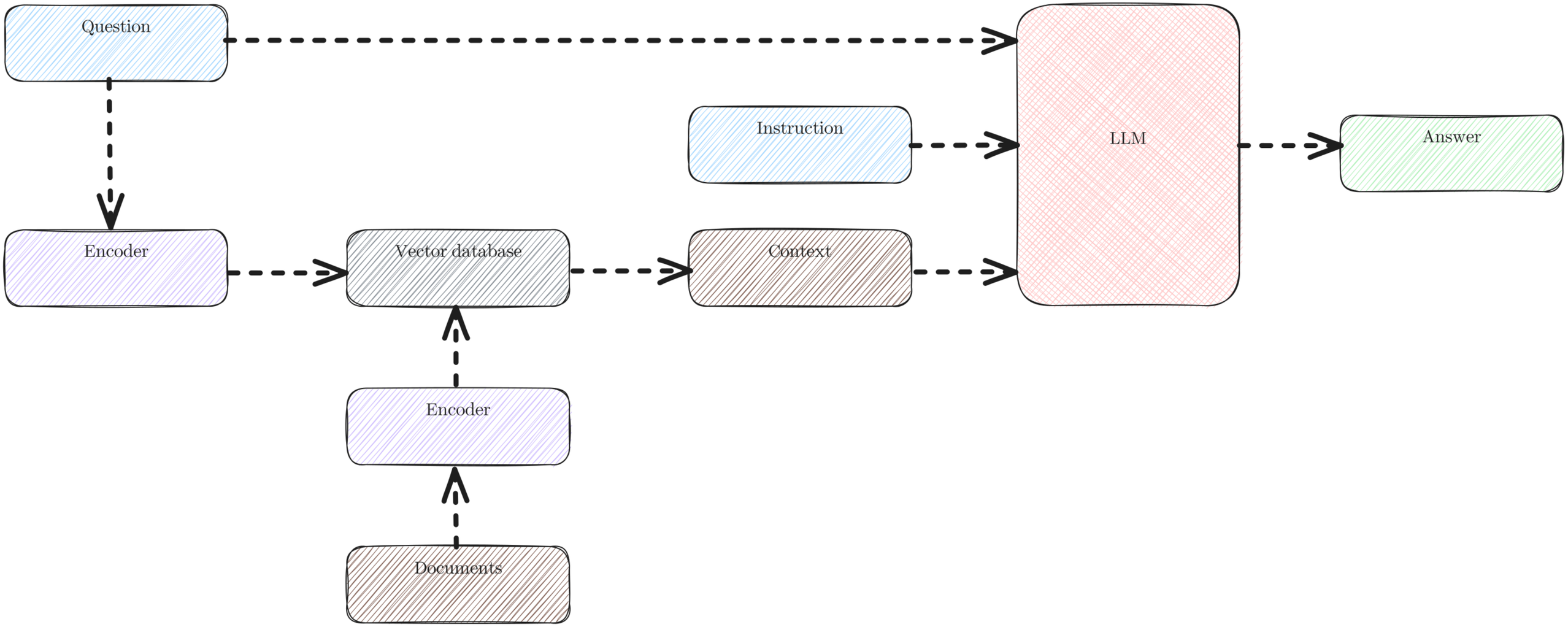
https://en.wikipedia.org/wiki/Prompt_engineering

# RAG

- combines retrieval (fetching) and generation
- searches a knowledge base for relevant info
- uses fetched data from “enterprise” databases to generate responses

https://en.wikipedia.org/wiki/Prompt_engineering

# RAG


Question
Encoder
Vector database
Encoder
Documents
Instruction
Context
LLM
Answer

# LLMs

- ChatGPT
- Claude
- Llama
- Mistral

# LLAMA — the new standard

- open-source model with licensing restrictions
- developed by Meta (Facebook)
- NLP tasks
- base for many fine-tuned models

https://llama.meta.com/

# GGUF (GPT-Generated Unified Format)

- a file format for AI models
- designed for efficient inference
- successor to GGML format
- introduced by llama.cpp project
- quickly adopted by open-source AI projects
- used by Mistral
- A self-contained parser for Java is available: Llama3.java

# ONNX

- Open Neural Network Exchange
- general purpose
- focus on interoperability
- open-source community project
- ONNX Java Runtime is available: https://onnxruntime.ai/docs/get-started/with-java.html

https://en.wikipedia.org/wiki/Open_Neural_Network_Exchange

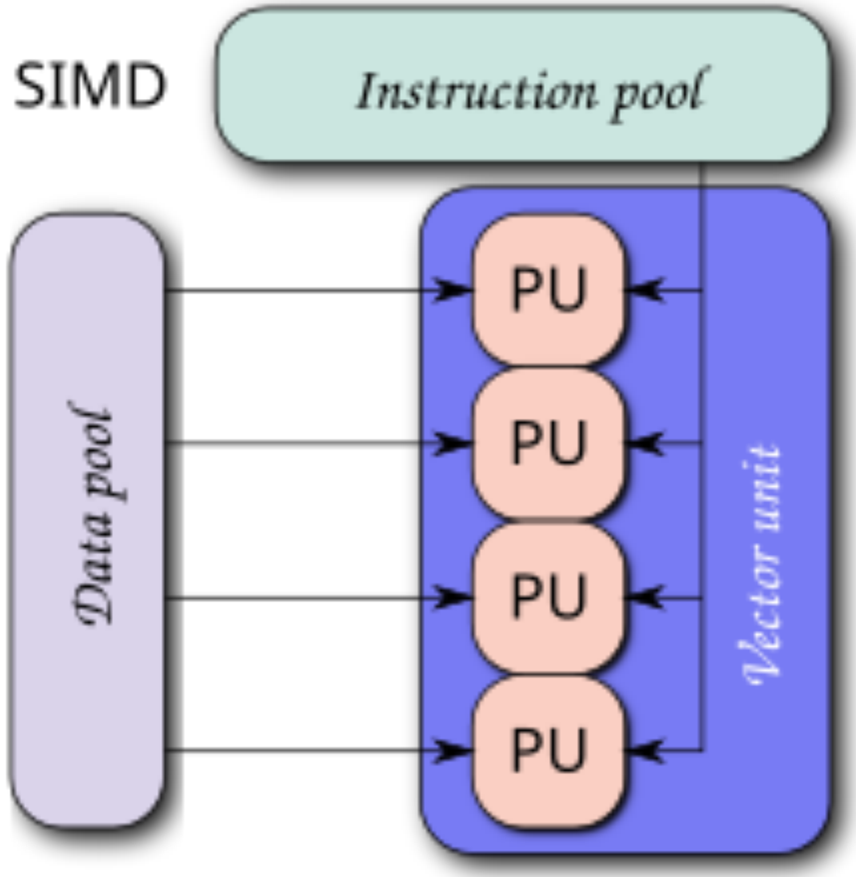
# SIMD

**Single instruction, multiple data** (**SIMD**) is a type of parallel processing in Flynn's taxonomy. SIMD can be internal (part of the hardware design) and it can be directly accessible through an instruction set architecture (ISA), but it should not be confused with an ISA. SIMD describes computers with multiple processing elements that perform the same operation on multiple data points simultaneously.

Such machines exploit data level parallelism, but not concurrency: there are simultaneous (parallel) computations, but each unit performs the exact same instruction at any given moment (just with different data). SIMD is particularly applicable to common tasks such as adjusting the contrast in a digital image or adjusting the volume of digital audio. Most modern CPU designs include SIMD instructions to improve the performance of multimedia use. SIMD has three different subcategories in Flynn's 1972 Taxonomy, one of which is SIMT. SIMT should not be confused with software threads or hardware threads, both of which are task time-sharing (time-slicing). SIMT is true simultaneous parallel hardware-level execution.

https://en.wikipedia.org/wiki/Single_instruction,_multiple_data

# SIMD




https://en.wikipedia.org/wiki/Single_instruction,_multiple_data

# Java Vector API

- (Clear and concise API)
- Platform agnostic
- Reliable runtime compilation and performance on x64 and AArch64:
  - Streaming SIMD Extensions
  - Advanced Vector Extensions (AVX)
  - ARM Aarch64: NEON and SVE

# Java Vector API

- Graceful degradation
- Alignment with Project Valhalla

# Java Vector API

- “efficient vector computations in Java”
- vector operations optimally compiled to vector instructions on CPU architectures
- in Java 23 is the eight preview. Vector API is available in Java 21 as preview
- Vector<E>, VectorSpecies<E>, VectorMask<E>, VectorShuffle<E>
- matrix vector multiplication is crucial in a neural networks

# DeepNetts

- full stack Java
- netbeans based IDE
- build ML models with Java
- creation and distribution of neural network models
- reference implementation for JSR-381 (Visual Recognition (VisRec) Specification): https://github.com/JavaVisRec/visrec-api/wiki/Getting-Started-Guide

https://www.deepnetts.com/

# LangChain4j

- similar to multiple EntityManagers behind a boundary
- llm abstraction similar to JPA
- fully integrated with MicroProfile / Quarkus
- works with GraalVM

# LangChain4j

- prompts
- chains
- agents and tools
- memory
- llm
- indexes
- DocumentSplitter, VectorDB, DocumentLoader

# Hexagonal Architectures

DEEP NETTS

bedrock

claude

langchain4j

java application

Meta

llama

jlama

llama3.java


ChatGPT

# Thank You!