CS 696 Applied Large Language Models
Spring Semester, 2025
Doc 17 News, Activation Steering, Performance Issues v2
Mar 20, 2025

Generative AI use surging among consumers for online shopping

Adobe Analytics platform

1 trillion visits to U.S. retail sites

Nov. 1 and Dec. 31,

traffic from generative AI sources increased by 1,300% compared to the year prior

February,

Traffic from generative AI sources increased by 1,200% compared to July 2024

But don’t give absolute numbers

0.001% -> 0.01% is a 1,000% increase

https://searchengineland.com/generative-ai-surging-online-shopping-report-453312

Startups are fastest growing

Y Combinator CEO Garry Tan

For about a quarter of the current YC startups, 95% of the code was written by AI

“…you don’t need a team of 50 or 100 engineers,”

For the last nine months, the entire batch of YC companies in aggregate grew 10% per week, he said.

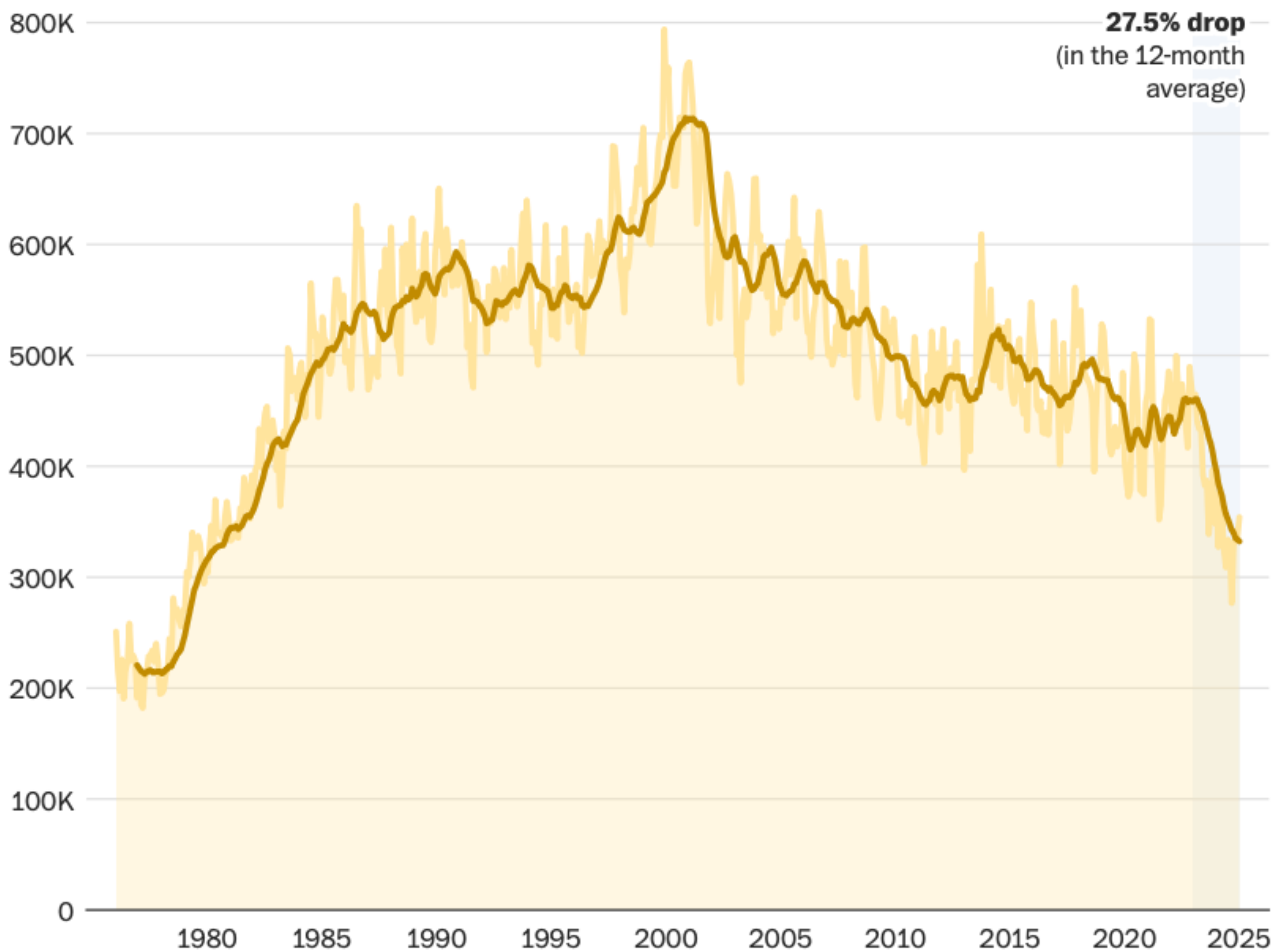
https://www.cnbc.com/2025/03/15/y-combinator-startups-are-fastest-growing-in-fund-history-because-of-ai.html

More than a quarter of computer-programming jobs just vanished

What happened?

https://www.washingtonpost.com/business/2025/03/14/programming-jobs-lost-artificial-intelligence/

March 14, 2025



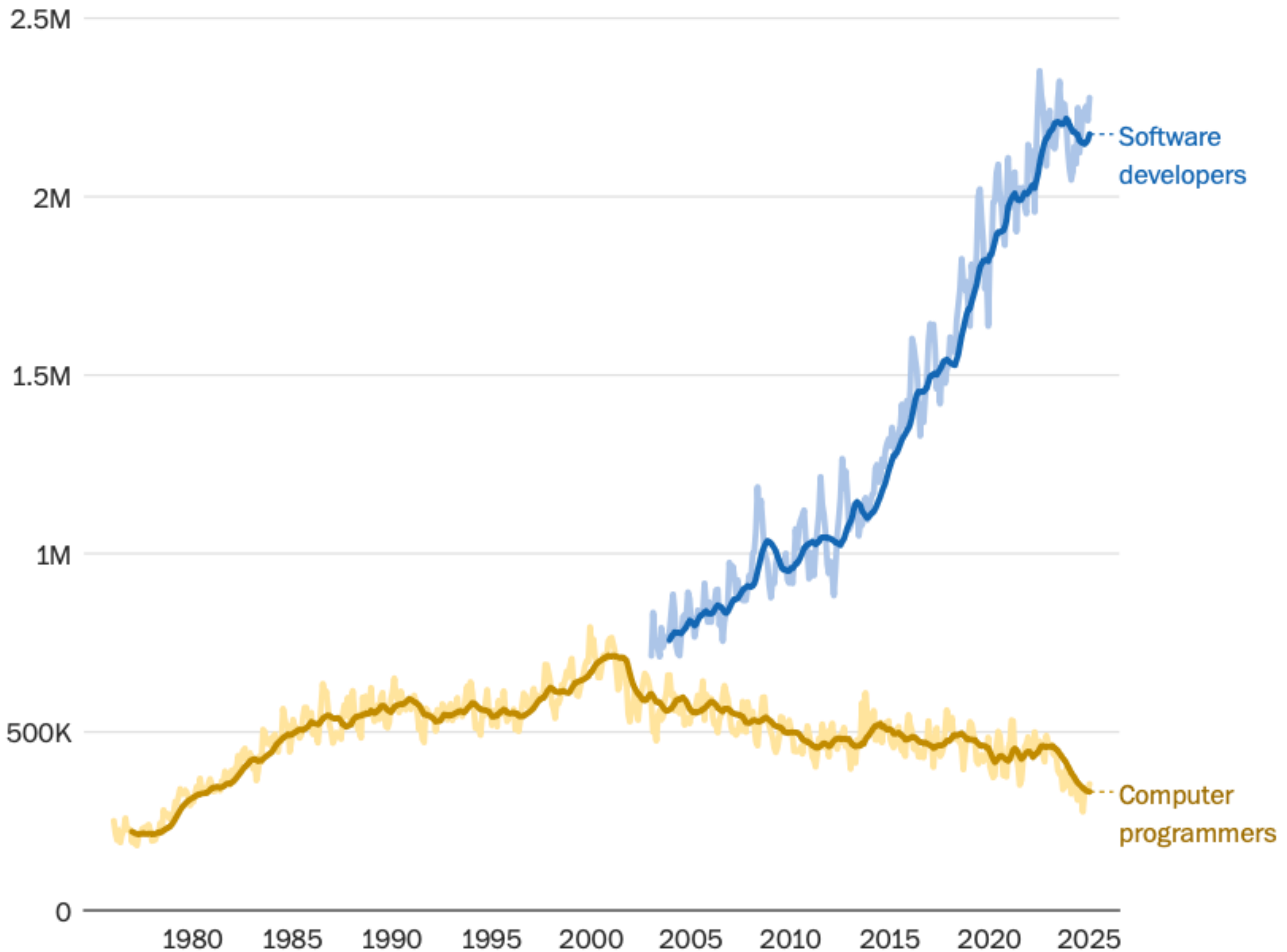
Note: Dark line shows 12-month average

Source: Current Population Survey from the Bureau of Labor Statistics via IPUMS

DEPARTMENT OF DATA / THE WASHINGTON POST

The government began tracking software developers in 2003

Total U.S. employment for select occupations

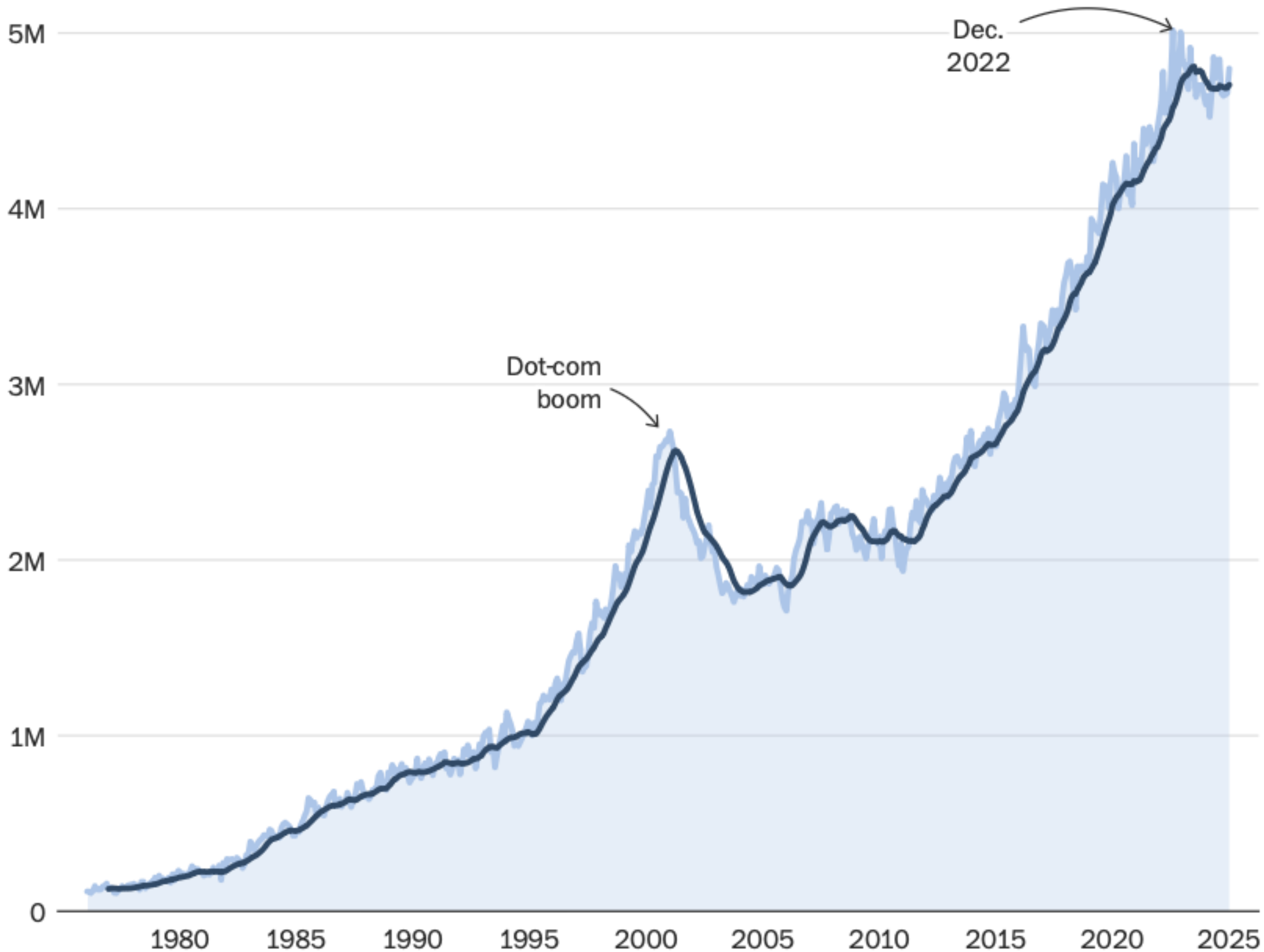


Note: Dark line shows 12-month average

Source: Current Population Survey from the Bureau of Labor Statistics via IPUMS

DEPARTMENT OF DATA / THE WASHINGTON POST

Employment in the computer and data-processing services industry



Note: Dark line shows 12-month average

Source: Source: Current Population Survey from the Bureau of Labor Statistics via IPUMS

DEPARTMENT OF DATA / THE WASHINGTON POST

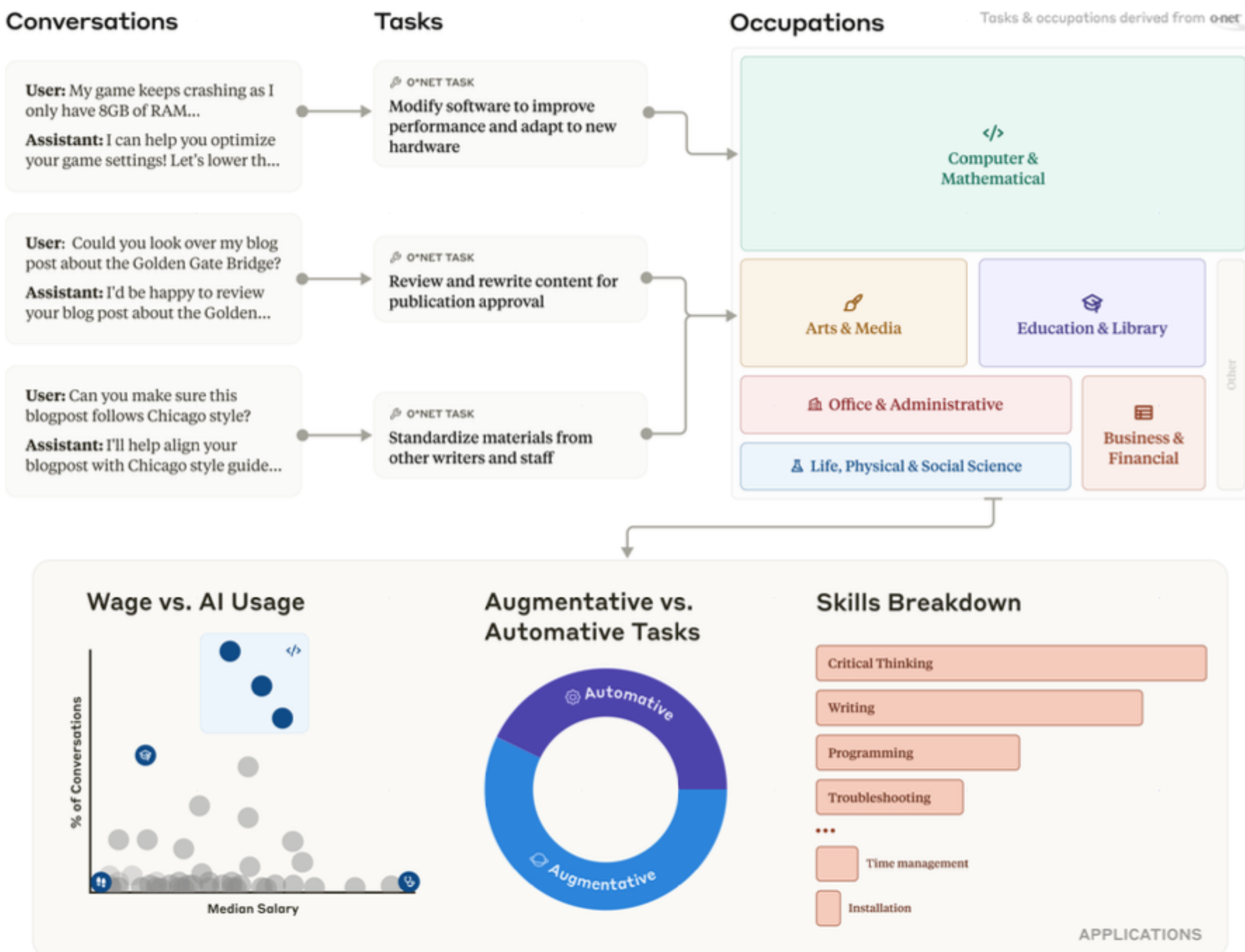
Which Economic Tasks are Performed with AI? Evidence from Millions of Claude Conversations

Four million Claude.ai conversations

December 2024 and January 2025

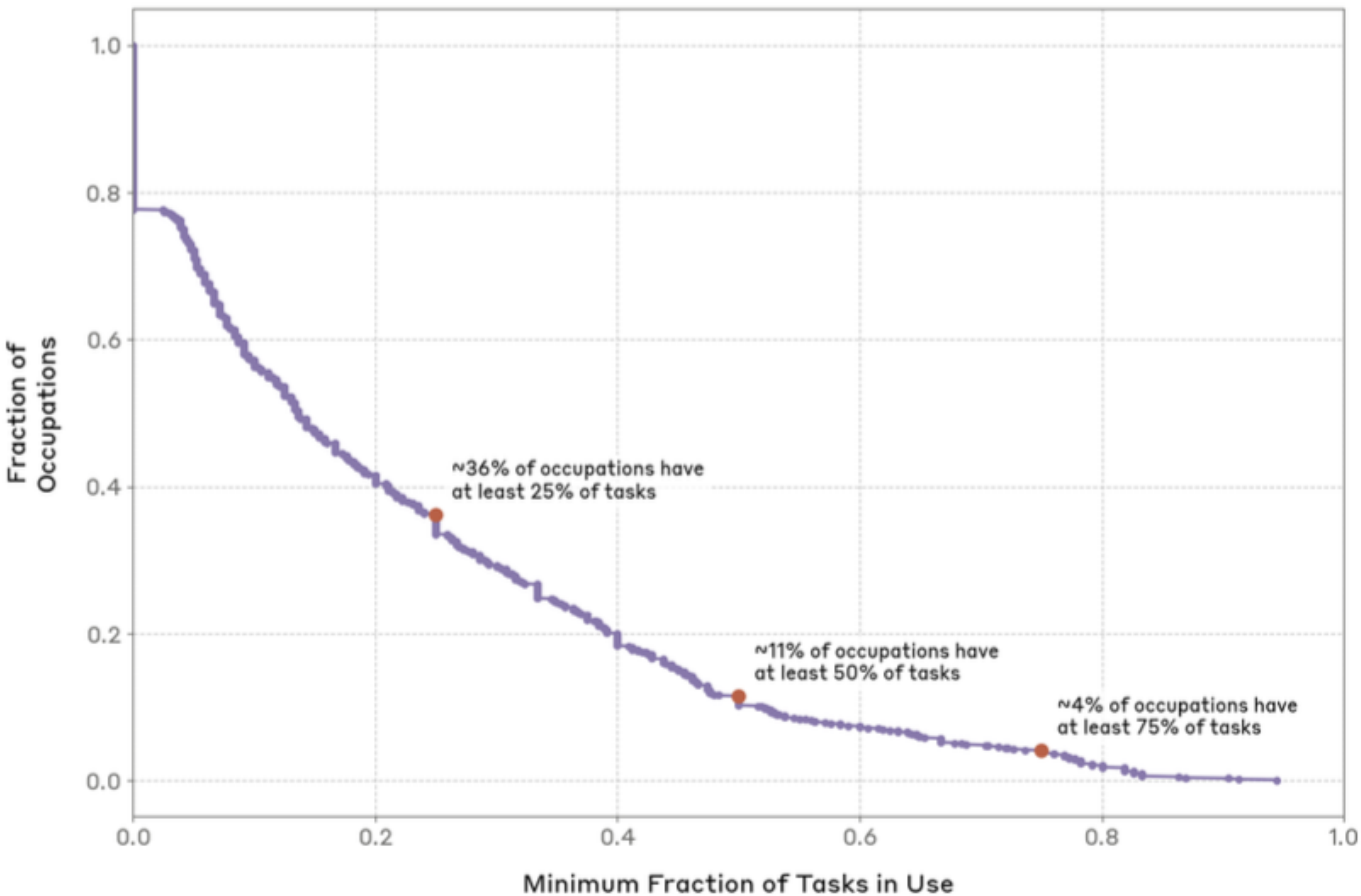
Tasks and occupations

https://assets.anthropic.com/m/2e23255f1e84ca97/original/Economic_Tasks_AI_Paper.pdf

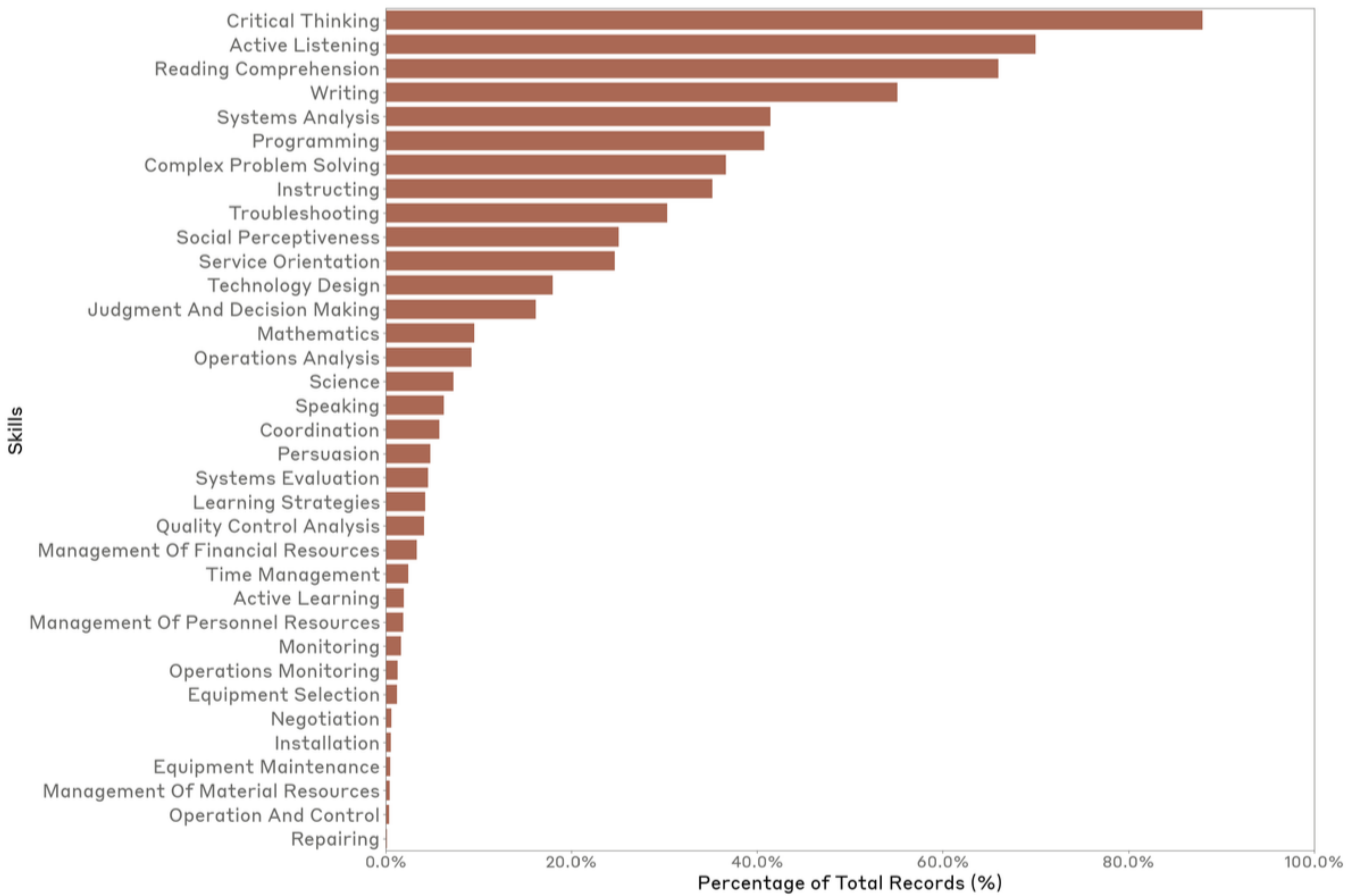


Conversations
Tasks
Occupations
Tasks & occupations derived from o*net
User: My game keeps crashing as I only have 8GB of RAM...
Assistant: I can help you optimize your game settings! Let's lower th...
O*NET TASK
Modify software to improve performance and adapt to new hardware
User: Could you look over my blog post about the Golden Gate Bridge?
Assistant: I'd be happy to review your blog post about the Golden...
O*NET TASK
Review and rewrite content for publication approval
User: Can you make sure this blogpost follows Chicago style?
Assistant: I'll help align your blogpost with Chicago style guide...
O*NET TASK
Standardize materials from other writers and staff
Computer & Mathematical
Arts & Media
Education & Library
Office & Administrative
Business & Financial
Life, Physical & Social Science
Other
Wage vs. AI Usage
% of Conversations
Median Salary
Augmentative vs. Automative Tasks
Automative
Augmentative
Skills Breakdown
Critical Thinking
Writing
Programming
Troubleshooting
...
Time management
Installation
APPLICATIONS

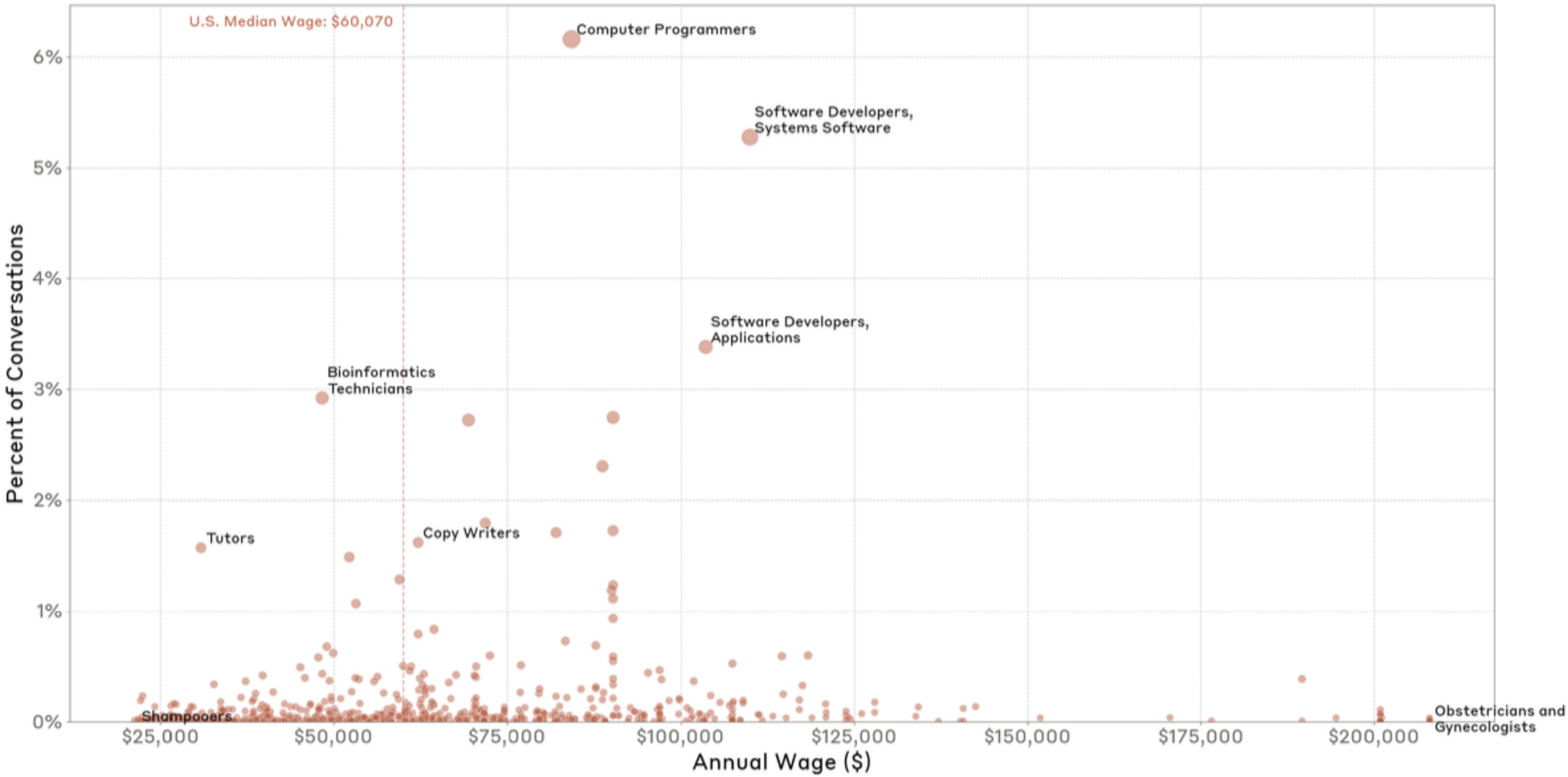
Depth of AI usage across occupations



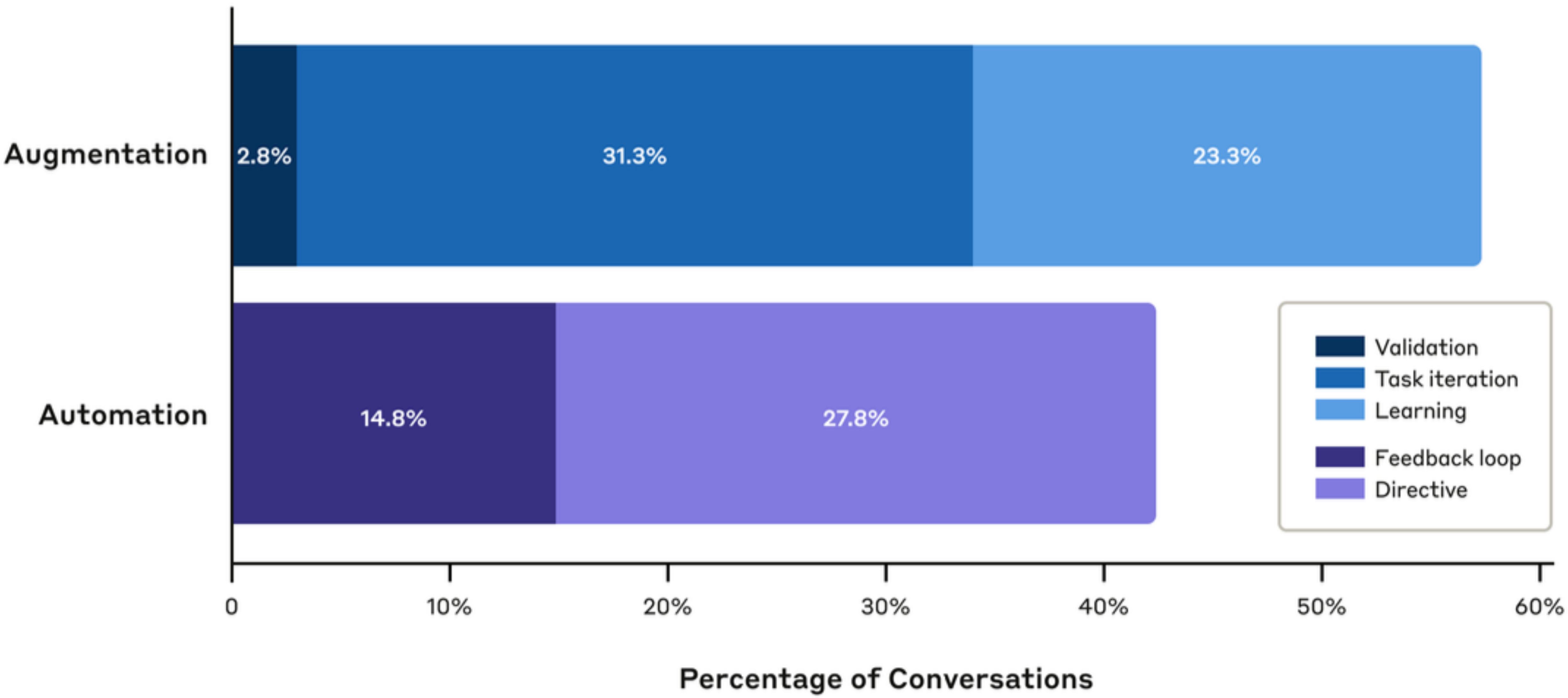
Distribution of occupational skills



Occupational usage of Claude.ai by annual wage



Augmentation vs Automation



Where are facts stored in LLMs

https://medium.com/@nikhilanandnj/where-are-facts-stored-in-large-language-models-0869914cfcbf

Locating and Editing Factual Associations in GPT

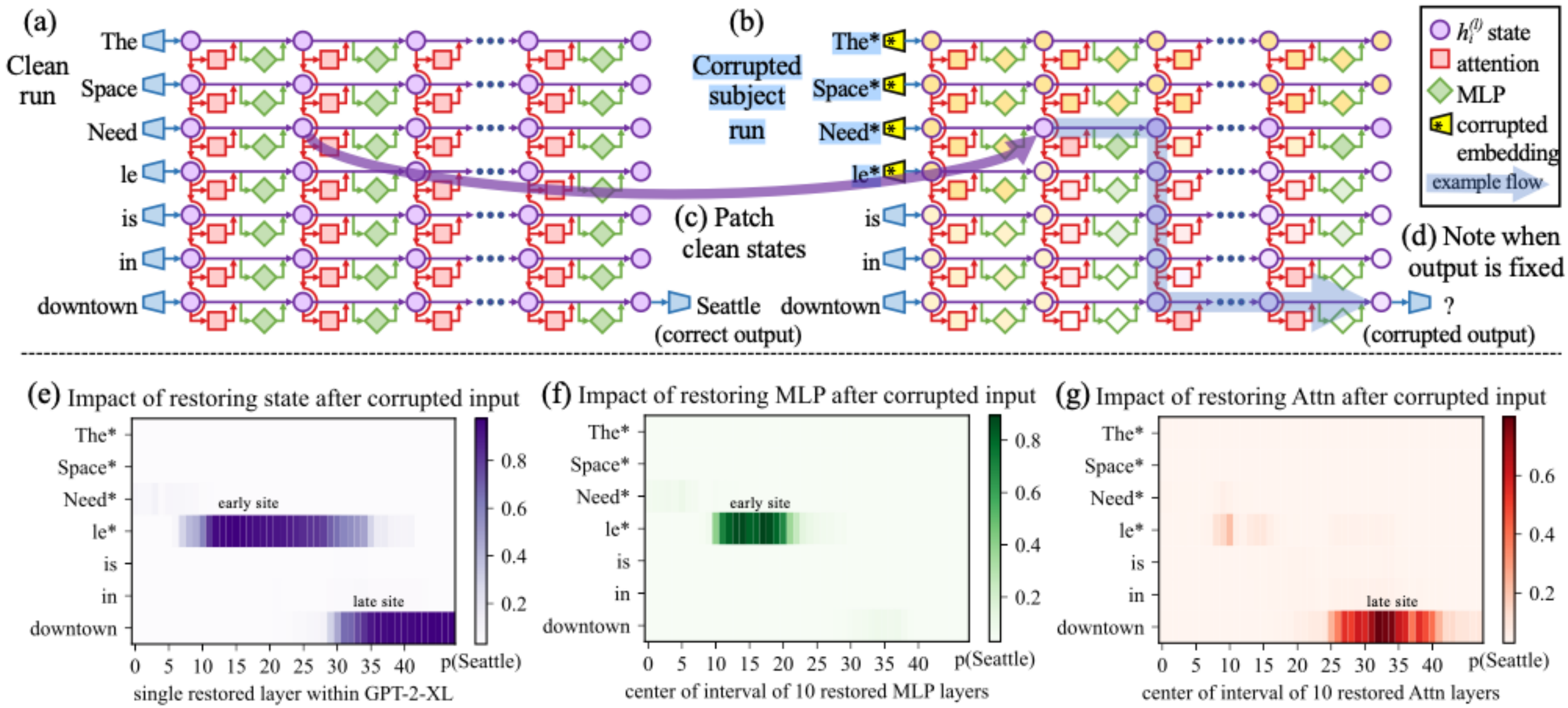
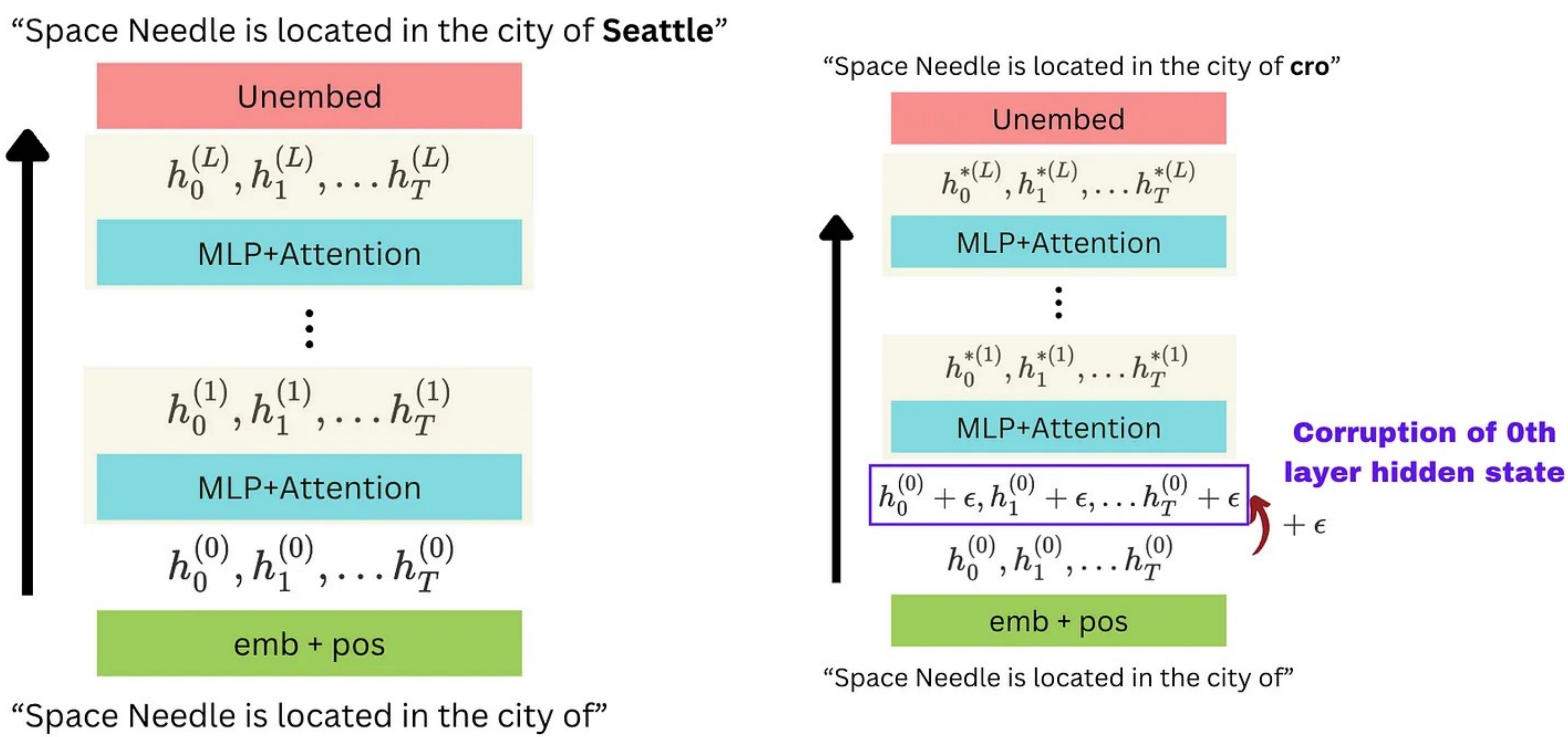
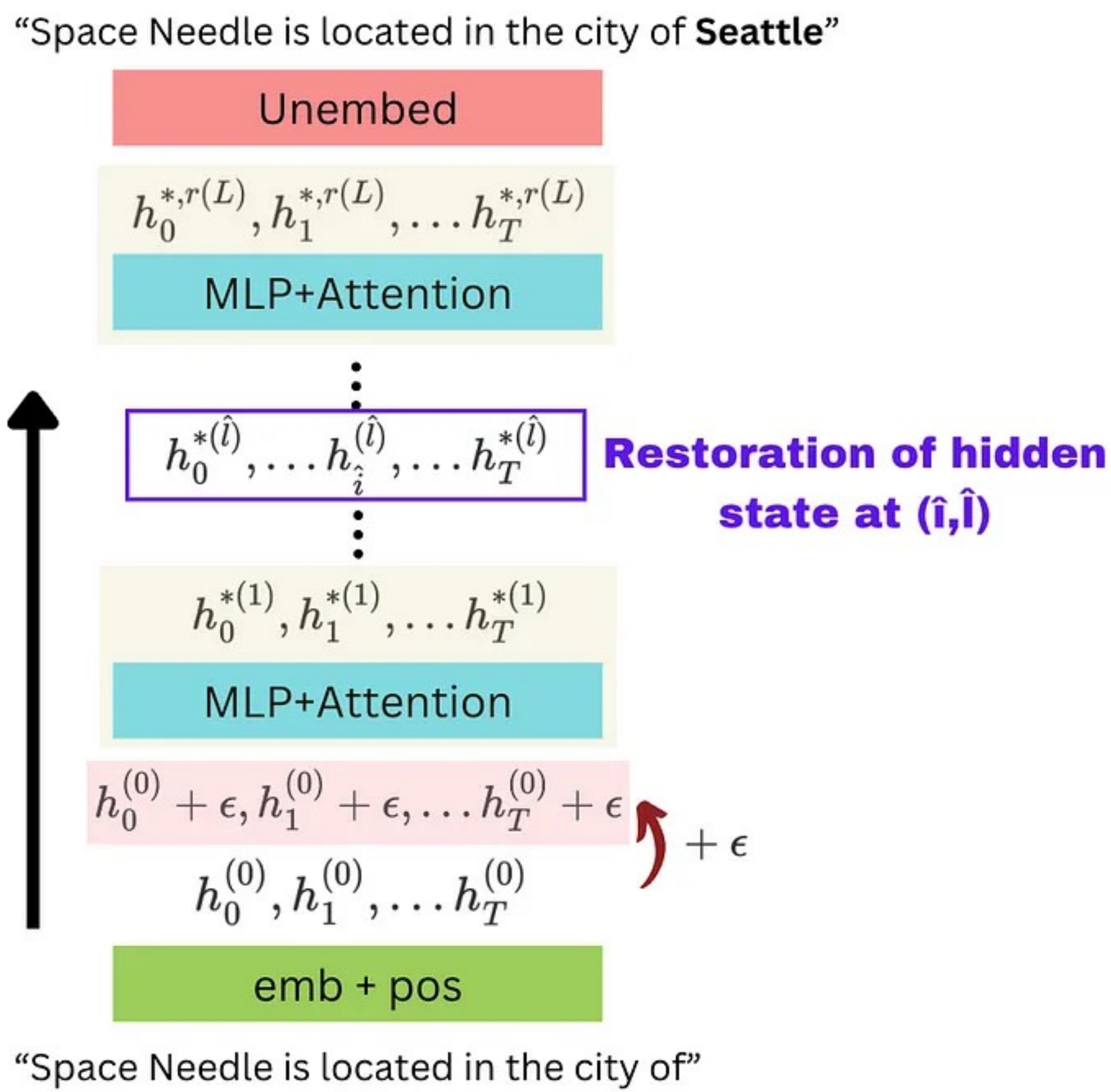


Figure 1: **Causal Traces** compute the causal effect of neuron activations by running the network twice: (a) once normally, and (b) once where we corrupt the subject token and then (c) restore selected internal activations to their clean value. (d) Some sets of activations cause the output to return to the original prediction; the light blue path shows an example of information flow. The causal impact on output probability is mapped for the effect of (e) each hidden state on the prediction, (f) only MLP activations, and (g) only attention activations.

STEERING LANGUAGE MODELS WITH ACTIVATION ENGINEERING



“Space Needle is located in the city of **Seattle**”
Unembed
$h_0^{(L)}, h_1^{(L)}, \ldots h_T^{(L)}$
MLP+Attention
$h_0^{(1)}, h_1^{(1)}, \ldots h_T^{(1)}$
MLP+Attention
$h_0^{(0)}, h_1^{(0)}, \ldots h_T^{(0)}$
emb + pos
“Space Needle is located in the city of”
“Space Needle is located in the city of **cro**”
Unembed
$h_0^{*(L)}, h_1^{*(L)}, \ldots h_T^{*(L)}$
MLP+Attention
$h_0^{*(1)}, h_1^{*(1)}, \ldots h_T^{*(1)}$
MLP+Attention
$h_0^{(0)} + \epsilon, h_1^{(0)} + \epsilon, \ldots h_T^{(0)} + \epsilon$
Corruption of 0th layer hidden state
$+ \epsilon$
$h_0^{(0)}, h_1^{(0)}, \ldots h_T^{(0)}$
emb + pos
“Space Needle is located in the city of”



MLP layers had a more significant causal effect than attention layers

Total Effect

$$o = \text{“Seattle”}$$

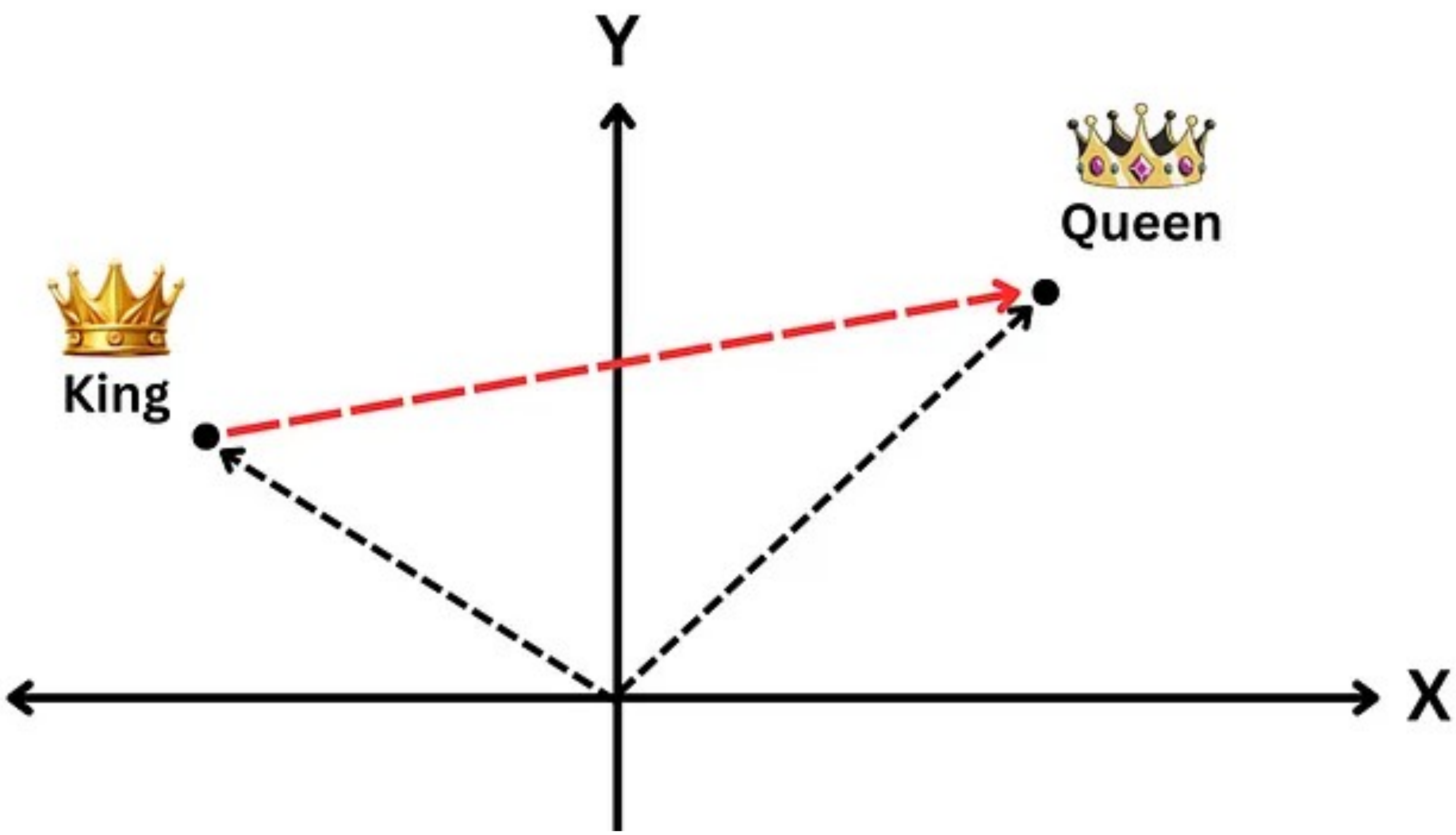
$$\mathrm{TE} = \mathbb{P}_*[o] - \mathbb{P}[o]$$

Indirect Effect

$$\mathrm{IE} = \mathbb{P}_{*,\text{clean } h_i^{(l)}}[o] - \mathbb{P}_*[o]$$

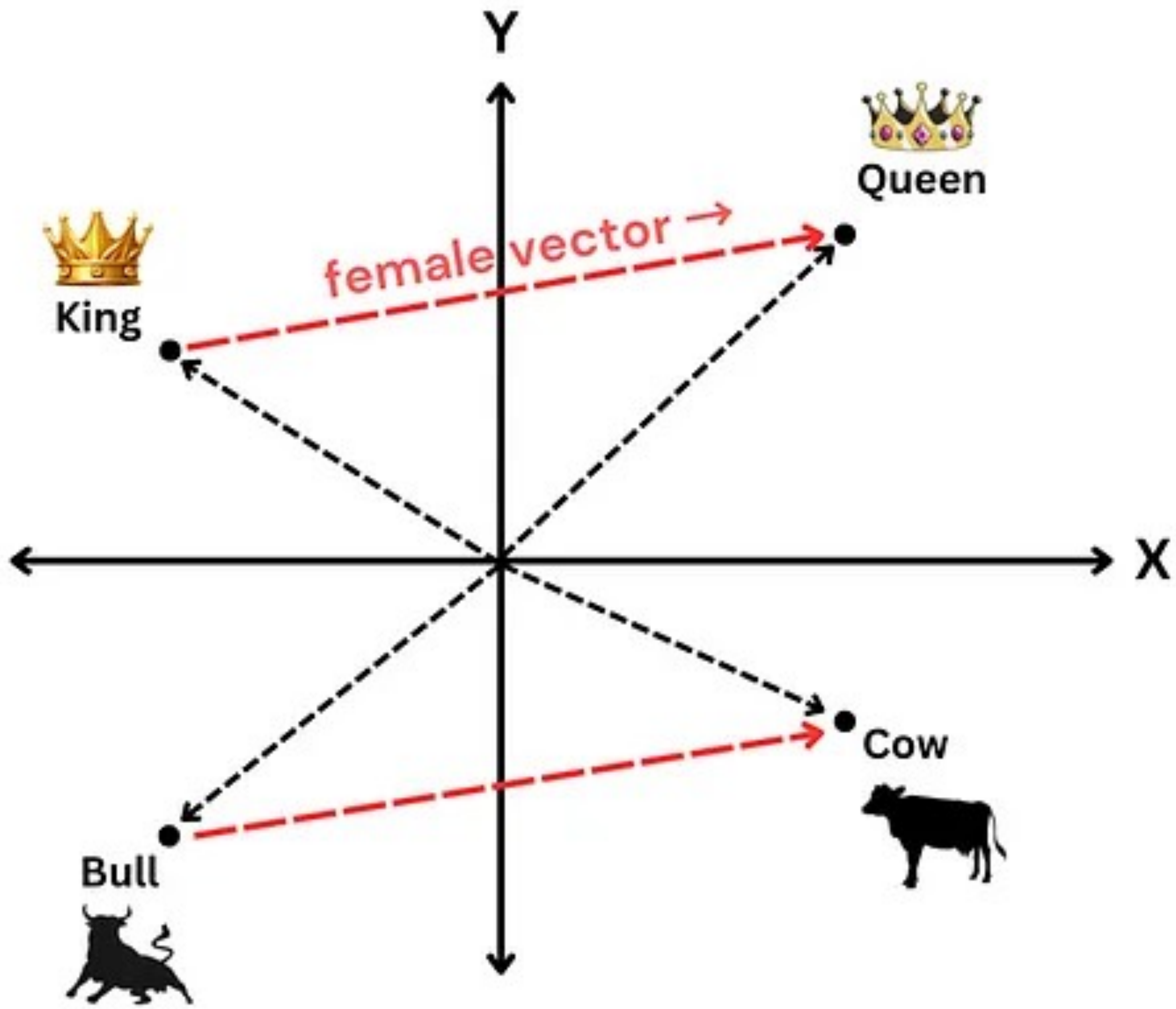
Activation Steering

Control and guide LLM outputs by modifying neuron activations



Understanding “steering” in LLMs
https://ai.gopubby.com/understanding-steering-in-llms-96faf6e0bee7

Activation Steering



Understanding “steering” in LLMs
https://ai.gopubby.com/understanding-steering-in-llms-96faf6e0bee7

Activation Steering - Finding Concept Vector

Collect Activation Data

Select a set of prompts that strongly exhibit the concept.

Example: Positivity:

"Describe a beautiful day."

"Tell me something inspiring."

"What makes people happy?"

Select a contrasting set of prompts that do not exhibit the concept.

Example: Neutral or negative:

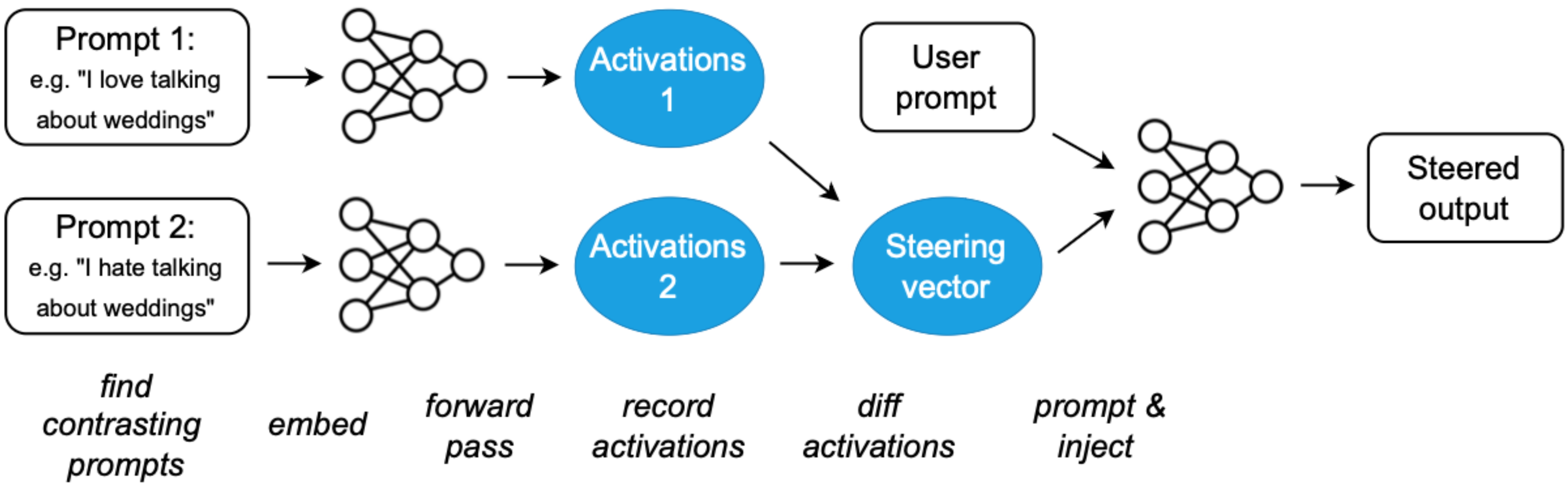
"What are common problems in life?"

"Describe a tragic event."

"What makes people sad?"

Pass these prompts through the model

Extract the hidden layer activations at a specific layer.



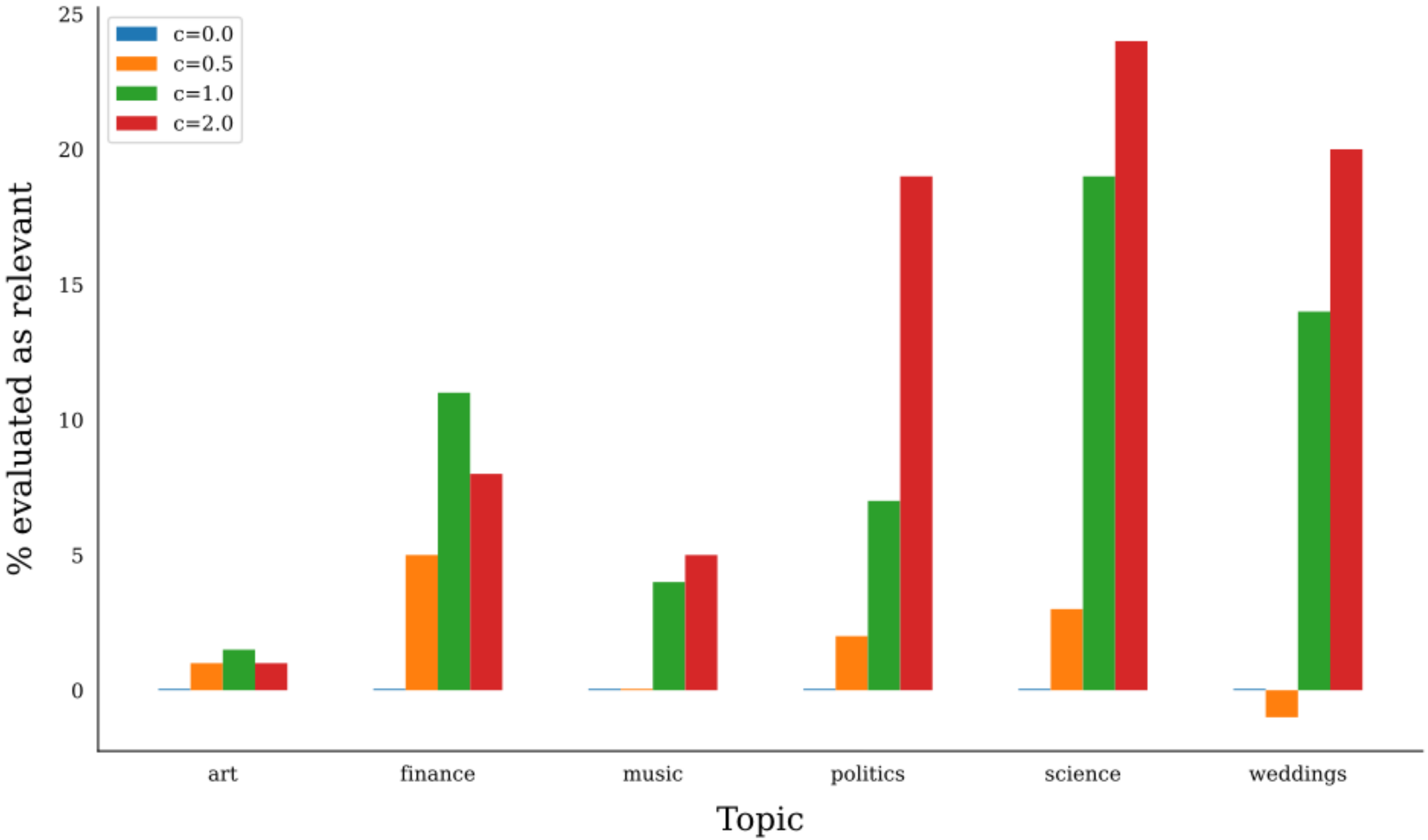
STEERING LANGUAGE MODELS WITH ACTIVATION ENGINEERING

Algorithm 1 ActAdd, optimization-free activation addition

Input: (p_+, p_-) = steering prompt pair, tokenized
p^* = user prompt
l = target layer
c = injection coefficient
a = sequence position to align $\mathbf{h}_A$ and $\mathbf{h}_{p^*}$
M = pretrained language model
Output: S = steered output

$(p'_+, p'_-) \leftarrow \texttt{pad\_right\_same\_token\_len}(p_+, p_-)$
$\mathbf{h}^l_+ \leftarrow M\texttt{.forward}(p'_+)\texttt{.activations}[l]$
$\mathbf{h}^l_- \leftarrow M\texttt{.forward}(p'_-)\texttt{.activations}[l]$
$\mathbf{h}^l_A \leftarrow \mathbf{h}^l_+ - \mathbf{h}^l_-$
$\mathbf{h}^l \leftarrow M\texttt{.forward}(p^*)\texttt{.activations}[l]$
$S \leftarrow M\texttt{.continue\_forward}(c\,\mathbf{h}^l_A + \mathbf{h}^l @ a)$

GPT-3.5 Boost in Relavance



A Sober Look at Steering Vectors for LLMs

by Joschka Braun, Dmitrii Krasheninnikov, Usman Anwar, RobertKirk,
Daniel Tan, David Scott Krueger

LESSWRONG, Nov 23, 2004

Current steering methods have substantial limitations

- Many steering methods
  - unreliable
  - often fail to generalize outside their specific training setup

- Steerability of different concepts varies significantly

- Typically used performance metrics overestimate steering effectiveness
  - Evaluated in artificial settings

- Methods are not compared on the same benchmarks and metrics

Mayo Clinic’s secret weapon against AI hallucinations

Reverse RAG in action

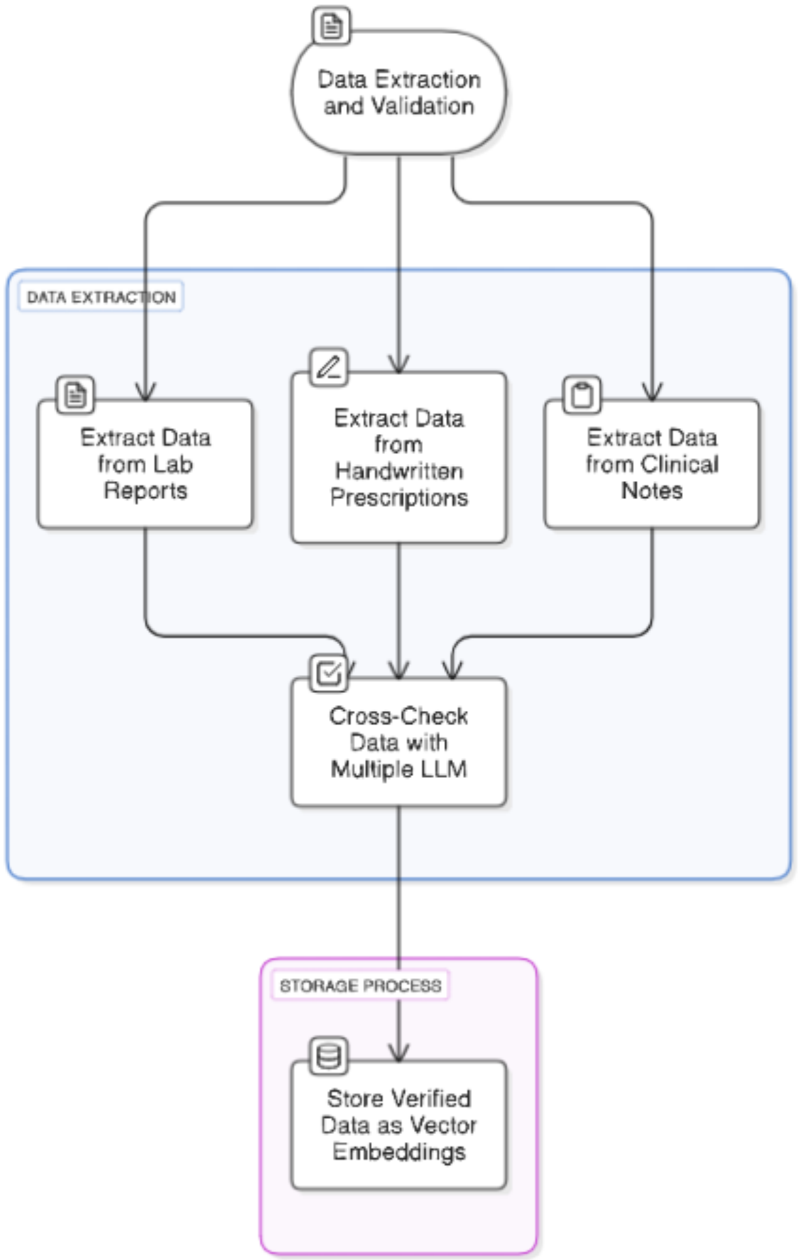
“The hospital has employed what is essentially backwards RAG, where the model extracts relevant information, then links every data point back to its original source content.

Remarkably, this has eliminated nearly all data-retrieval-based hallucinations in non-diagnostic use cases — allowing Mayo to push the model out across its clinical practice.”

https://venturebeat.com/ai/mayo-clinic-secret-weapon-against-ai-hallucinations-reverse-rag-in-action/

Mayo Clinic’s secret weapon against AI hallucinations

Reverse RAG in action



https://usmanshaheen.wordpress.com/2025/03/14/reverse-rag-reduce-hallucinations-and-errors-in-medical-genai-part-1/

Mayo Clinic’s secret weapon against AI hallucinations

Reverse RAG in action

Data Extraction

- LLM reads patients’ records
- Produces a summary or a list of facts

Fact Splitting

- Output split into individual data points

Source Matching

- AI is asked: “Where did this piece of information come from?”

Verification

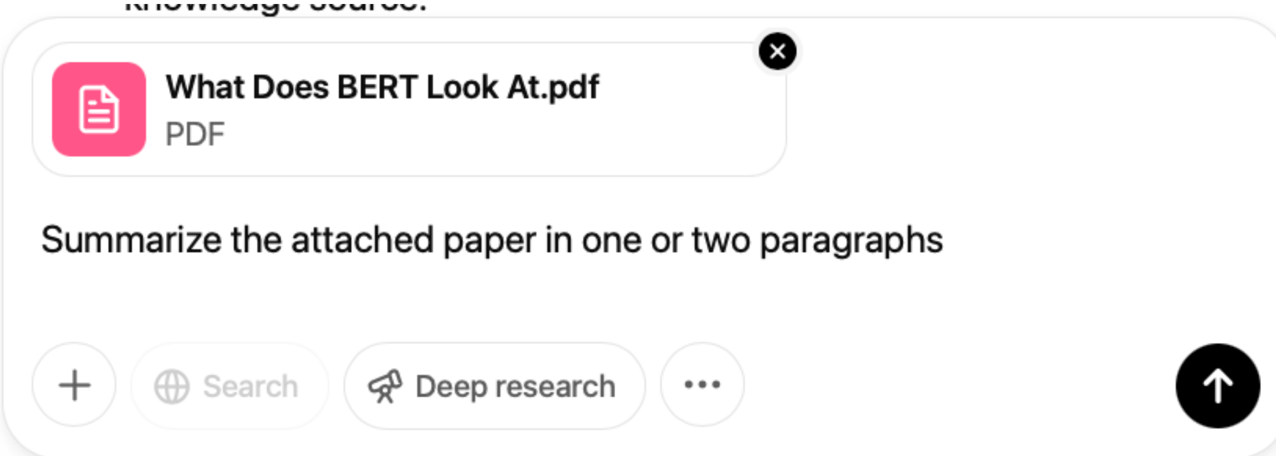
- A second LLM then compares each fact to the source
- Scores how well they align
- Looked for a causal relationship

Output with References

- Only facts with solid support are kept

https://usmanshaheen.wordpress.com/2025/03/14/reverse-rag-reduce-hallucinations-and-errors-in-medical-genai-part-1/

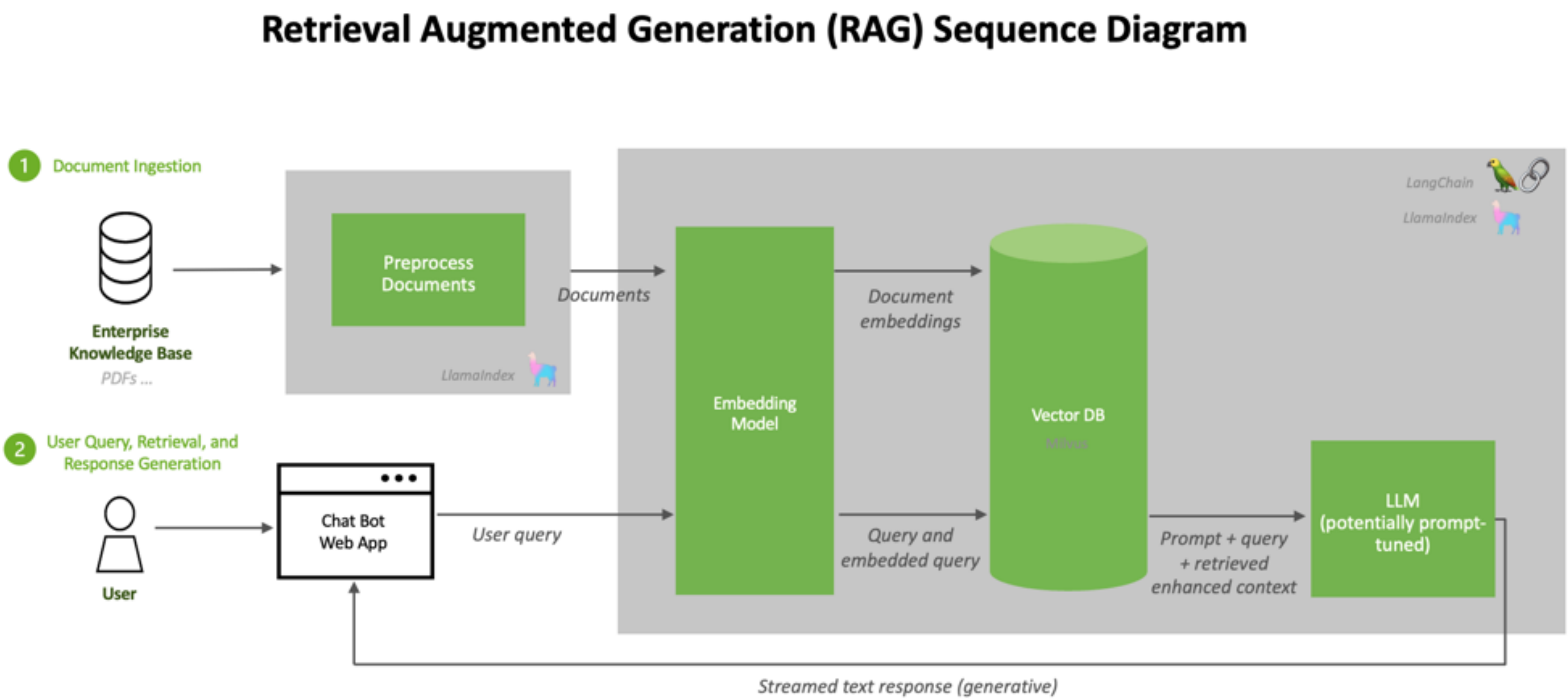
Retrieval-Augmented Generation (RAG)



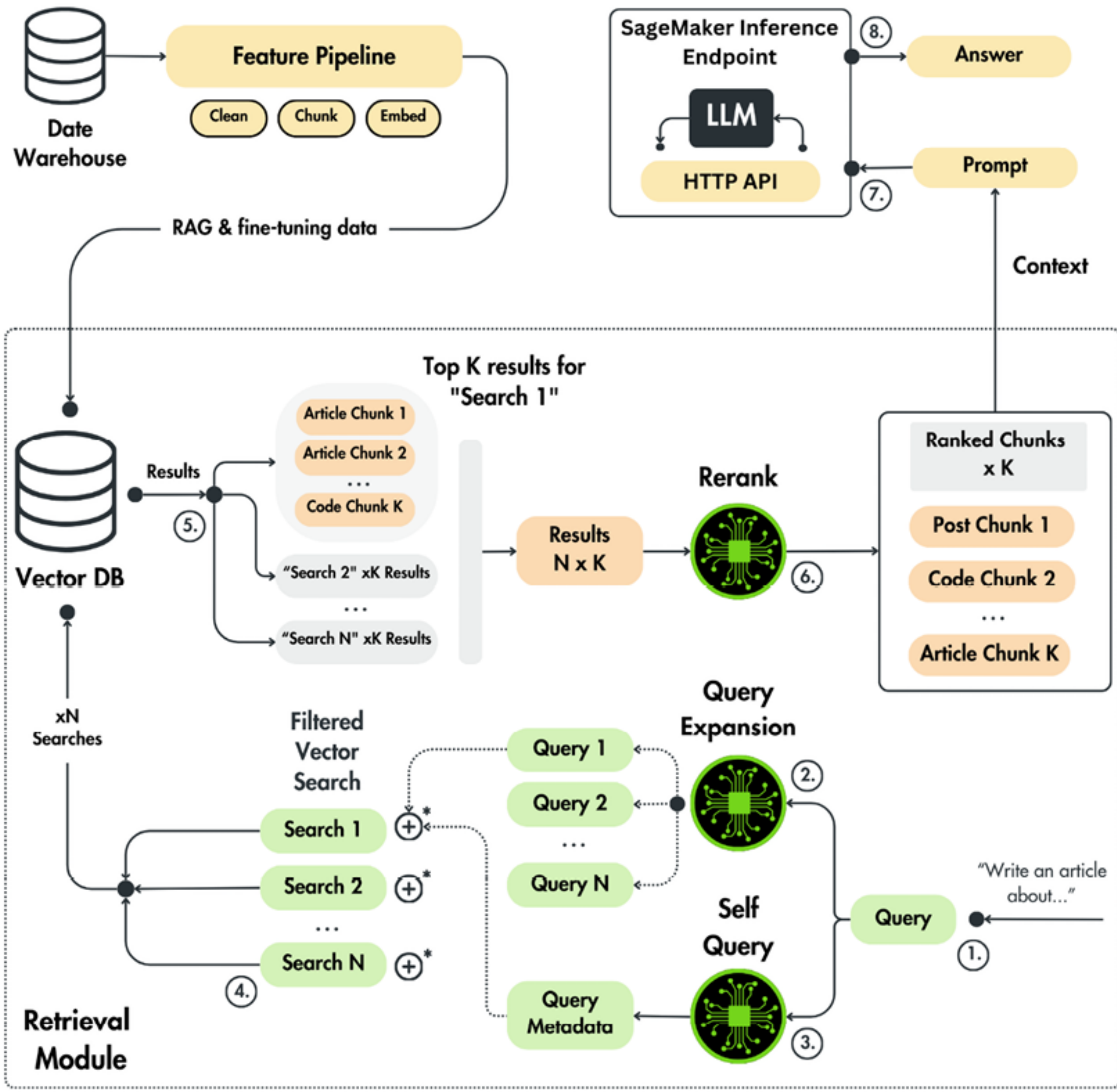
The paper "What Does BERT Look At? An Analysis of BERT's Attention" by Kevin Clark, Urvashi Khandelwal, Omer Levy, and Christopher D. Manning explores the attention mechanisms of BERT to understand what linguistic structures it learns during self-supervised training. The study analyzes attention heads in BERT to identify patterns in how they attend to different tokens in input sequences. Key findings reveal that certain attention heads specialize in specific linguistic roles, such as attending to direct objects of verbs, determiners of nouns, or coreferent mentions, often with high accuracy. The authors also observe that BERT frequently attends to special tokens like [SEP], sometimes using them as a "no-op" when attention functions are not needed. Additionally, attention heads tend to cluster by behavior, with similar functions appearing in the same layers.

Retrieval-Augmented Generation (RAG)

Augment query with content from a document database



blob:https://chatgpt.com/7120a495-3da4-4d4b-9f2c-8b9b2145c41d


Date
Warehouse
Feature Pipeline
Clean
Chunk
Embed
RAG & fine-tuning data
SageMaker Inference
Endpoint
LLM
HTTP API
8.
Answer
Prompt
7.
Context
Top K results for
"Search 1"
Article Chunk 1
Article Chunk 2
Code Chunk K
Results
5.
Vector DB
"Search 2" xK Results
"Search N" xK Results
Results
N x K
Rerank
6.
Ranked Chunks
x K
Post Chunk 1
Code Chunk 2
Article Chunk K
Query
Expansion
Query 1
Query 2
Query N
2.
xN
Searches
Filtered
Vector
Search
Search 1
Search 2
Search N
4.
"Write an article
about..."
Query
1.
Self
Query
Query
Metadata
3.
Retrieval
Module

Embedding and Vector Database

Goal

Find documents that are relevant to a query

Issue

Textual search is not good at finding relevant documents

Divide document into chunks

Convert chunks into embedded vectors

Store each embedded vector with link to document

Searching for Similar Vectors

Euclidean Distance

```
def euclidean_distance(vec1, vec2):
    return np.linalg.norm(vec1 - vec2)
```

Dot Product

Cosine distance

```
def cosine_distance(vec1,vec2):
    cosine = 1 - abs((np.dot(vec1,vec2)/(
        np.linalg.norm(vec1)*np.linalg.norm(vec2))))
    return cosine
```

Unlocking Data with Generative AI and RAG, Keith Bourne

Searching for Similar Vectors

S1 = 'This blanket has such a cozy temperature for me!',
S2 ='I am so much warmer and snug using this spread!',
S3='Taylor Swift was 34 years old in 2024.'

Embed them as a vector

	Euclidean Distance	Dot Product	Cosine Distance
S1 & S2	4.6	12.3	0.45
S1 & S3	7.3	-0.8	0.97
S2 & S3	6.3	0.9	0.95

Unlocking Data with Generative AI and RAG, Keith Bourne

SentenceTransformer

https://www.sbert.net/index.html

```
from sentence_transformers import SentenceTransformer

# 1. Load a pretrained Sentence Transformer model
model = SentenceTransformer("all-MiniLM-L6-v2")

# The sentences to encode
sentences = [
    "The weather is lovely today.",
    "It's so sunny outside!",
    "He drove to the stadium.",
]

# 2. Calculate embeddings by calling model.encode()
embeddings = model.encode(sentences)
print(embeddings.shape)
# [3, 384]

# 3. Calculate the embedding similarities
similarities = model.similarity(embeddings, embeddings)
print(similarities)
```

```
tensor([[1.0000, 0.6660, 0.1046],
        [0.6660, 1.0000, 0.1411],
        [0.1046, 0.1411, 1.0000]])
```

SentenceTransformer - Models

Original Models

Semantic Search Models

```
from sentence_transformers import SentenceTransformer

model = SentenceTransformer("multi-qa-mpnet-base-cos-v1")

query_embedding = model.encode("How big is London")
passage_embeddings = model.encode([
    "London is known for its financial district",
    "London has 9,787,426 inhabitants at the 2011 census",
    "The United Kingdom is the fourth largest exporter of goods in the world",
])

similarity = model.similarity(query_embedding, passage_embeddings)
```

```
tensor([[0.4656, 0.6142, 0.2697]])
```

SentenceTransformer - Models

Multi-QA Models

Trained on 215M question-answer pairs from various sources and domains, including StackExchange, Yahoo Answers, Google & Bing search queries

Multilingual Models

ar, bg, ca, cs, da, de, el, en, es, et, fa, fi, fr, fr-ca, gl, gu, he, hi, hr, hu, hy, id, it, ja, ka, ko, ku, lt, lv, mk, mn, mr, ms, my, nb, nl, pl, pt, pt-br, ro, ru, sk, sl, sq, sr, sv, th, tr, uk, ur, vi, zh-cn, zh-tw.

Semantically similar sentences within one language or across languages

Joint Image & Text Embeddings

```
from sentence_transformers import SentenceTransformer, util
from PIL import Image
import glob
import torch
import pickle
import zipfile
from IPython.display import display
from IPython.display import Image as IPImage
import os
from tqdm.autonotebook import tqdm
torch.set_num_threads(4)
```

```
#First, we load the respective CLIP model
model = SentenceTransformer('clip-ViT-B-32')
```

Download Images

```
# Next, we get about 25k images from Unsplash
img_folder = 'photos/'
if not os.path.exists(img_folder) or len(os.listdir(img_folder)) == 0:
    os.makedirs(img_folder, exist_ok=True)

    photo_filename = 'unsplash-25k-photos.zip'
    if not os.path.exists(photo_filename):   #Download dataset if does not exist
        util.http_get('http://sbert.net/datasets/'+photo_filename, photo_filename)

    #Extract all images
    with zipfile.ZipFile(photo_filename, 'r') as zf:
        for member in tqdm(zf.infolist(), desc='Extracting'):
            zf.extract(member, img_folder)
```

Compute the embeddings

```
use_precomputed_embeddings = True

if use_precomputed_embeddings:
    emb_filename = 'unsplash-25k-photos-embeddings.pkl'
    if not os.path.exists(emb_filename):   #Download dataset if does not exist
        util.http_get('http://sbert.net/datasets/'+emb_filename, emb_filename)

    with open(emb_filename, 'rb') as fIn:
        img_names, img_emb = pickle.load(fIn)
    print("Images:", len(img_names))
else:
    img_names = list(glob.glob('unsplash/photos/*.jpg'))
    print("Images:", len(img_names))
    img_emb = model.encode([Image.open(filepath) for filepath in img_names], batch_size=128,
convert_to_tensor=True, show_progress_bar=True)
```

Search function

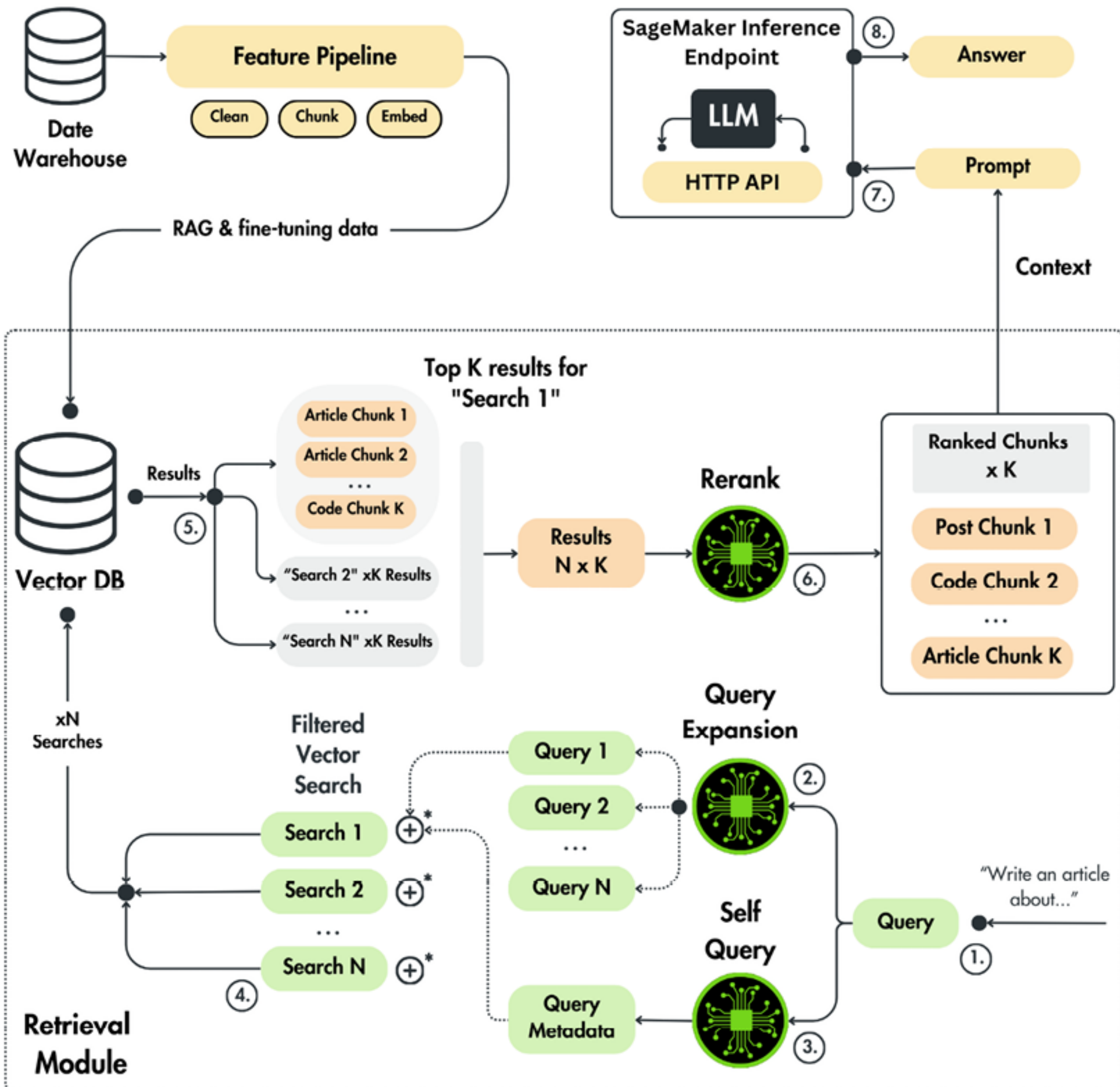
```
# Next, we define a search function.
def search(query, k=3):
    # First, we encode the query (which can either be an image or a text string)
    query_emb = model.encode([query], convert_to_tensor=True, show_progress_bar=False)

    # Then, we use the util.semantic_search function, which computes the cosine-similarity
    # between the query embedding and all image embeddings.
    # It then returns the top_k highest ranked images, which we output
    hits = util.semantic_search(query_emb, img_emb, top_k=k)[0]

    print("Query:")
    display(query)
    for hit in hits:
        print(img_names[hit['corpus_id']])
        display(IPImage(os.path.join(img_folder, img_names[hit['corpus_id']]), width=200))
```

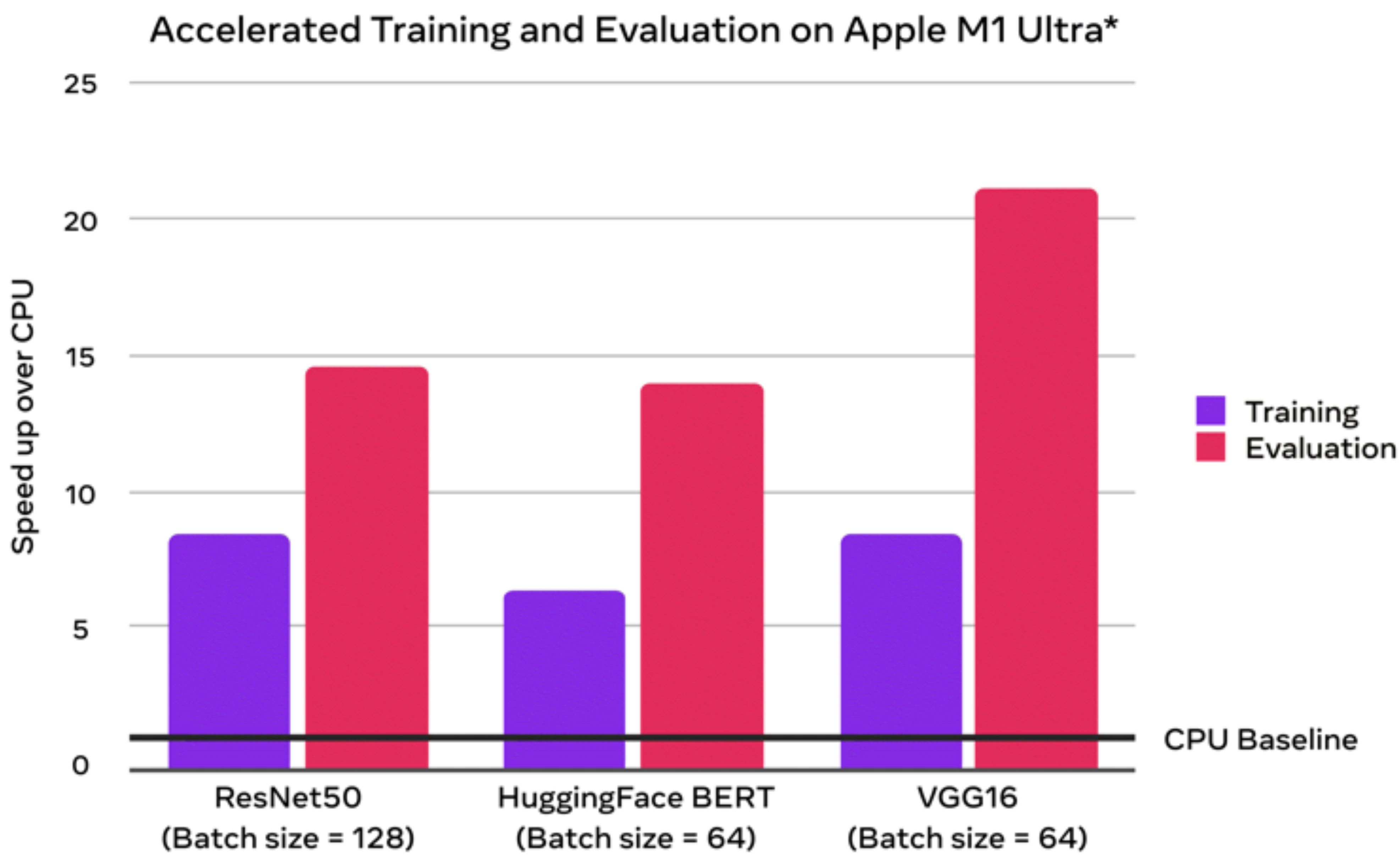
A Search

search("Two dogs playing in the snow")



Feature Pipeline
Clean
Chunk
Embed
Date Warehouse
RAG & fine-tuning data
SageMaker Inference Endpoint
LLM
HTTP API
8.
Answer
Prompt
7.
Context
Top K results for "Search 1"
Article Chunk 1
Article Chunk 2
Code Chunk K
Results
5.
Vector DB
"Search 2" xK Results
"Search N" xK Results
Results N x K
Rerank
6.
Ranked Chunks x K
Post Chunk 1
Code Chunk 2
Article Chunk K
xN Searches
Filtered Vector Search
Search 1
Search 2
Search N
4.
Query Expansion
Query 1
Query 2
Query N
2.
Self Query
Query Metadata
3.
Query
1.
"Write an article about..."
Retrieval Module

For Mac Users - MPS backend



For Mac Users - MPS backend

pip install torch torchvision torchaudio

```
# Check that MPS is available
if not torch.backends.mps.is_available():
    if not torch.backends.mps.is_built():
        print("MPS not available because the current PyTorch install was not "
              "built with MPS enabled.")
    else:
        print("MPS not available because the current MacOS version is not 12.3+ "
              "and/or you do not have an MPS-enabled device on this machine.")

else:
    mps_device = torch.device("mps")

    # Create a Tensor directly on the mps device
    x = torch.ones(5, device=mps_device)
    # Or
    x = torch.ones(5, device="mps")

    # Any operation happens on the GPU
    y = x * 2

    # Move your model to mps just like any other device
    model = YourFavoriteNet()
    model.to(mps_device)

    # Now every call runs on the GPU
    pred = model(x)
```

For Mac Users - MPS backend

Can only use 1 GPU

Some PyTorch operations are not implemented in MPS yet and will throw an error
Set the environment variable PYTORCH_ENABLE_MPS_FALLBACK=1

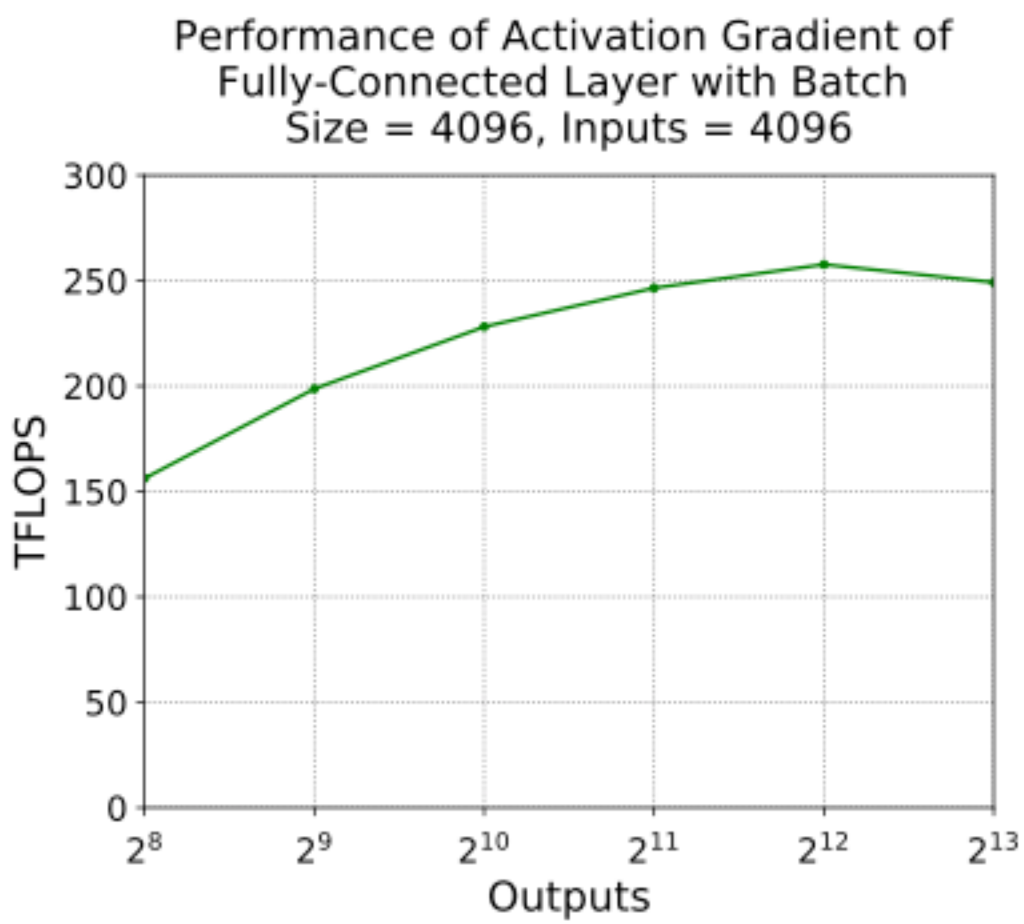
Huggingface Recommendations

Method/tool	Improves training speed	Optimizes memory utilization
Batch size choice	Yes	Yes
Gradient accumulation	No	Yes
Gradient checkpointing	No	Yes
Mixed precision training	Yes	Maybe*
torch_empty_cache_steps	No	Yes
Optimizer choice	Yes	Yes
Data preloading	Yes	No
DeepSpeed Zero	No	Yes
torch.compile	Yes	No
Parameter-Efficient Fine Tuning (PEFT)	No	Yes

Batch size & Layer Size

Batch sizes and input/output neuron counts use size 2^N.

Larger layers are more efficent to process

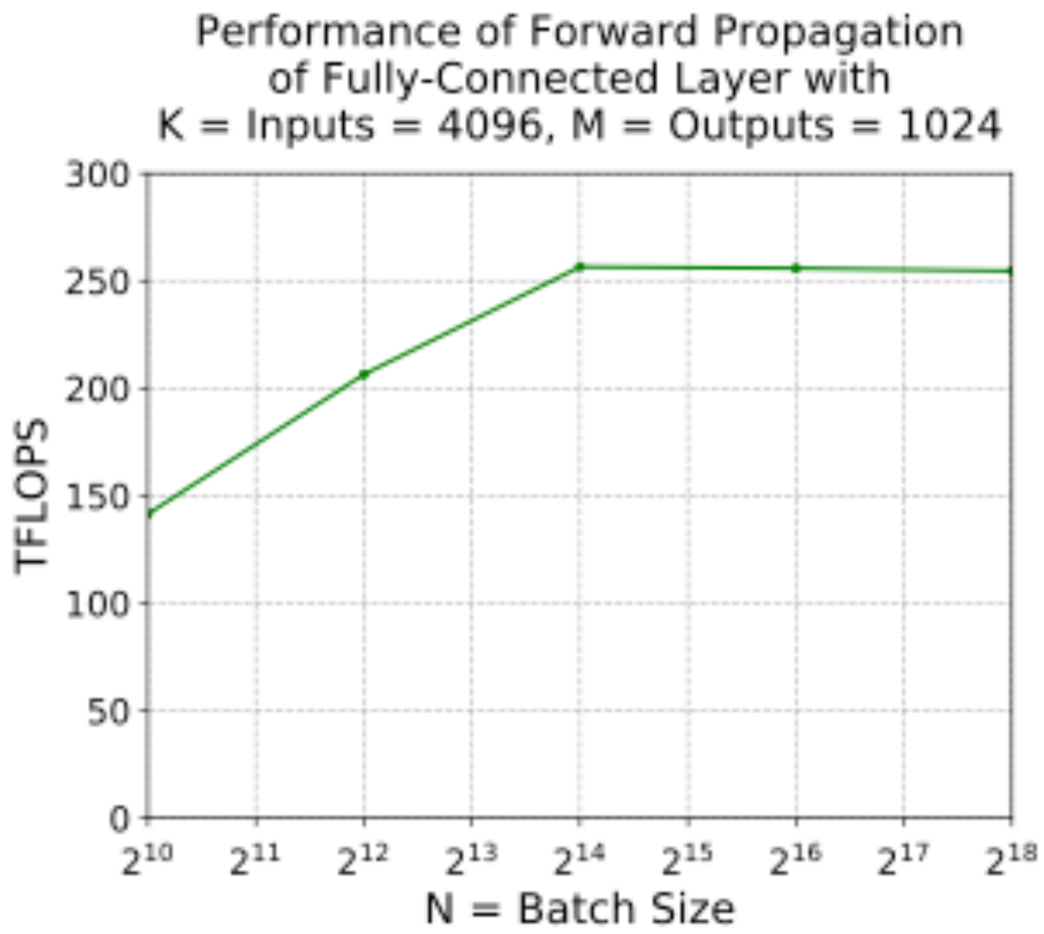


https://docs.nvidia.com/deeplearning/performance/dl-performance-fully-connected/index.html#input-features

Batch size & Layer Size

Batch sizes

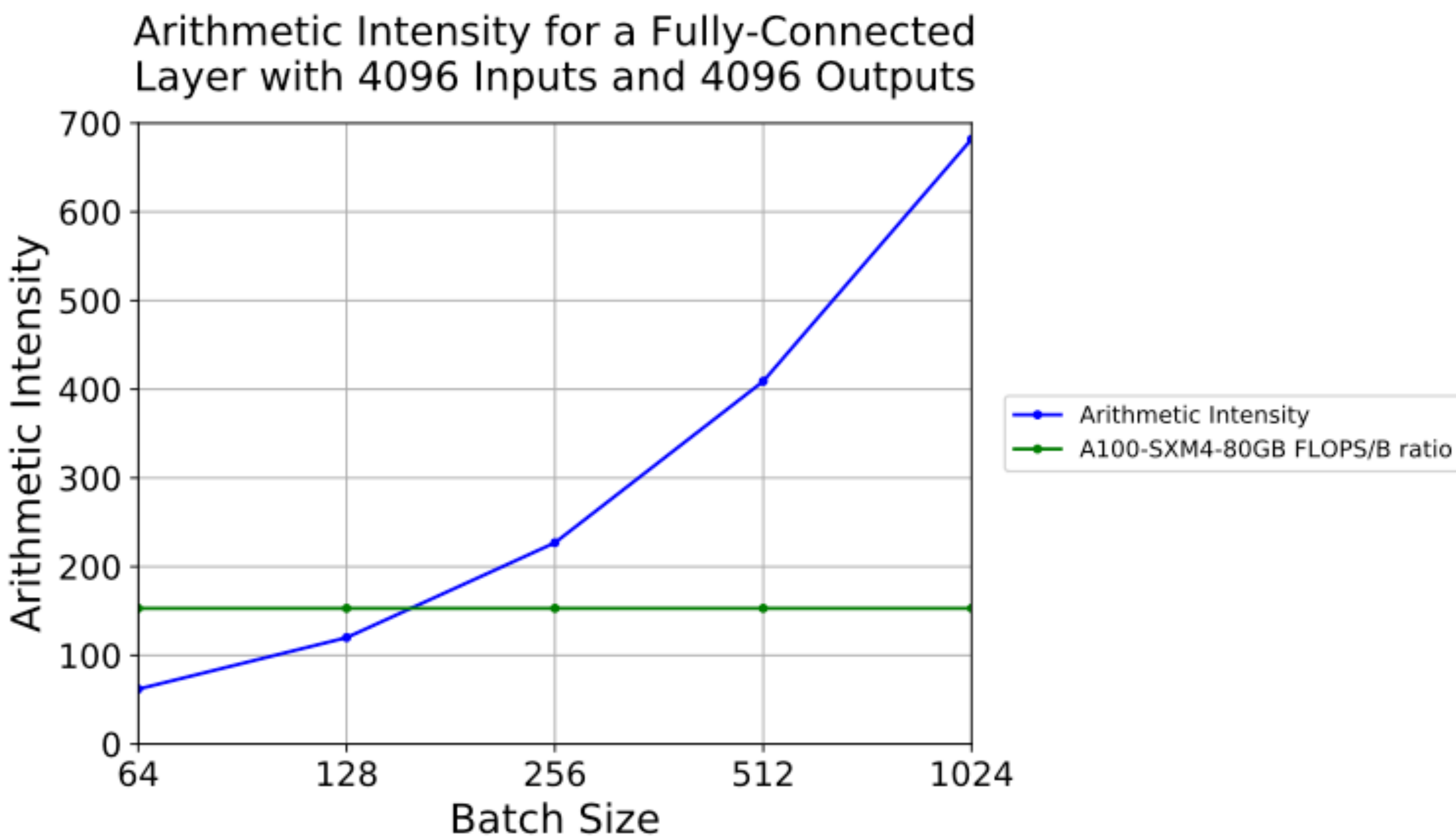
- Larger size more efficient
- Requires more memory



https://docs.nvidia.com/deeplearning/performance/dl-performance-fully-connected/index.html#input-features

Batch size & Layer Size

Batch sizes 128 and below are bandwidth limited on NVIDIA A100 accelerators.



https://docs.nvidia.com/deeplearning/performance/dl-performance-fully-connected/index.html#input-features

Gradient Accumulation

Calculate gradients in smaller increments due to memory constraints

```
training_args = TrainingArguments(
                    per_device_train_batch_size=1,
                    gradient_accumulation_steps=4, **default_args)
```

Gradient Checkpointing

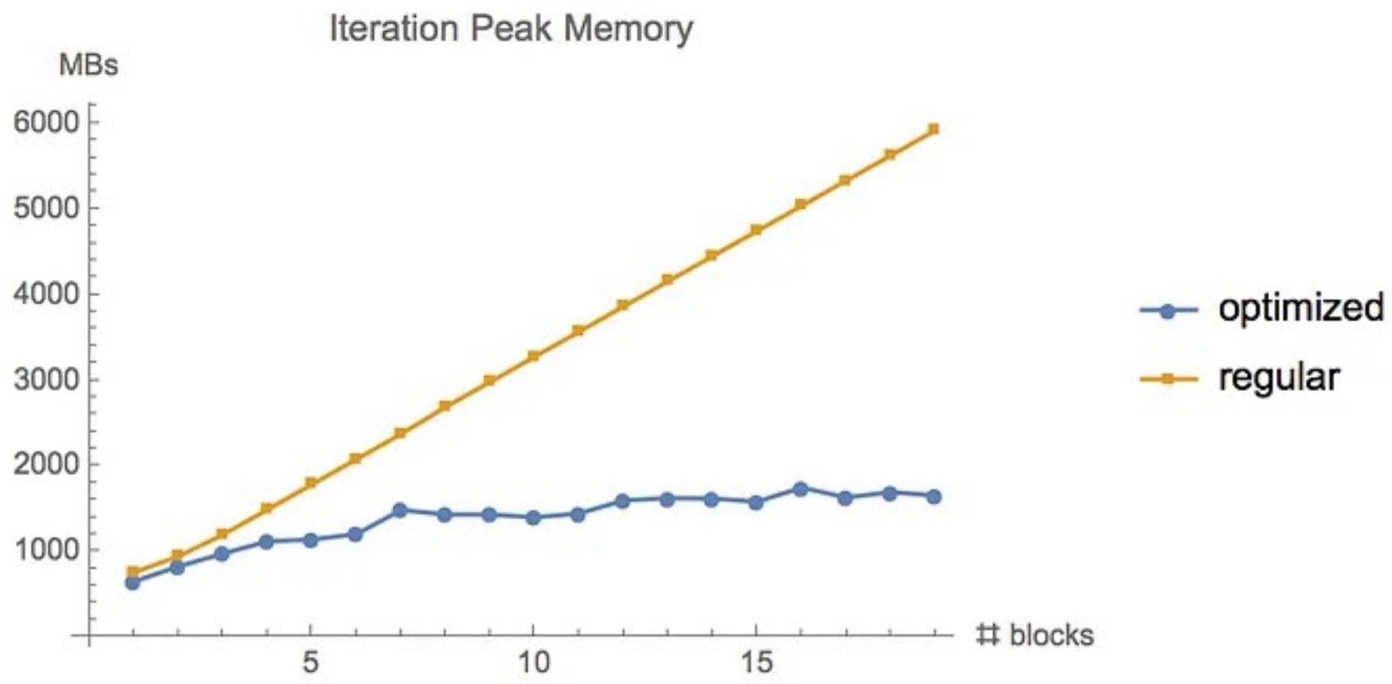
Activations from the forward pass consume a lot of memory

Deleting them and recomputing in the backward pass
- Saves memory but slows down backward pass

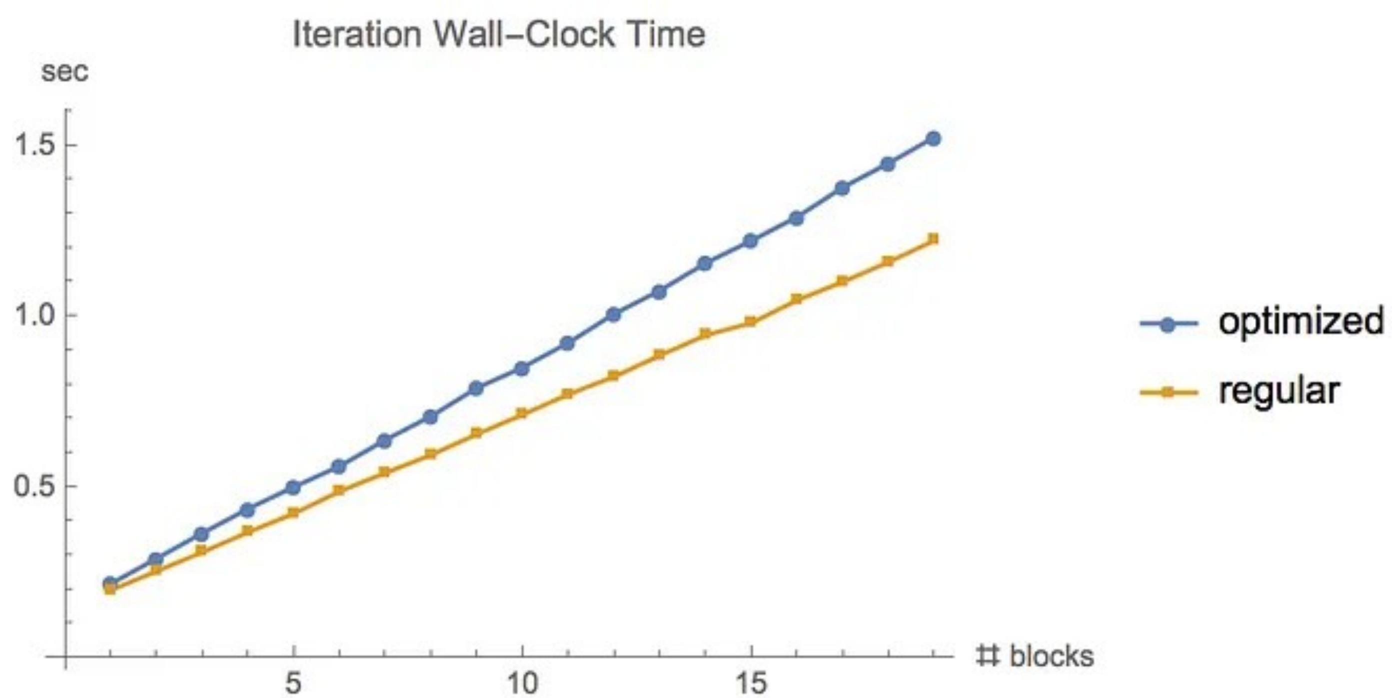
Gradient checkpointing
- Saves strategically selected activations
- Only a fraction of the activations need to be re-computed for the gradients.

```
training_args = TrainingArguments(
                    per_device_train_batch_size=1,
                    gradient_accumulation_steps=4,
                    gradient_checkpointing=True,
                    **default_args
)
```

Gradient Checkpointing

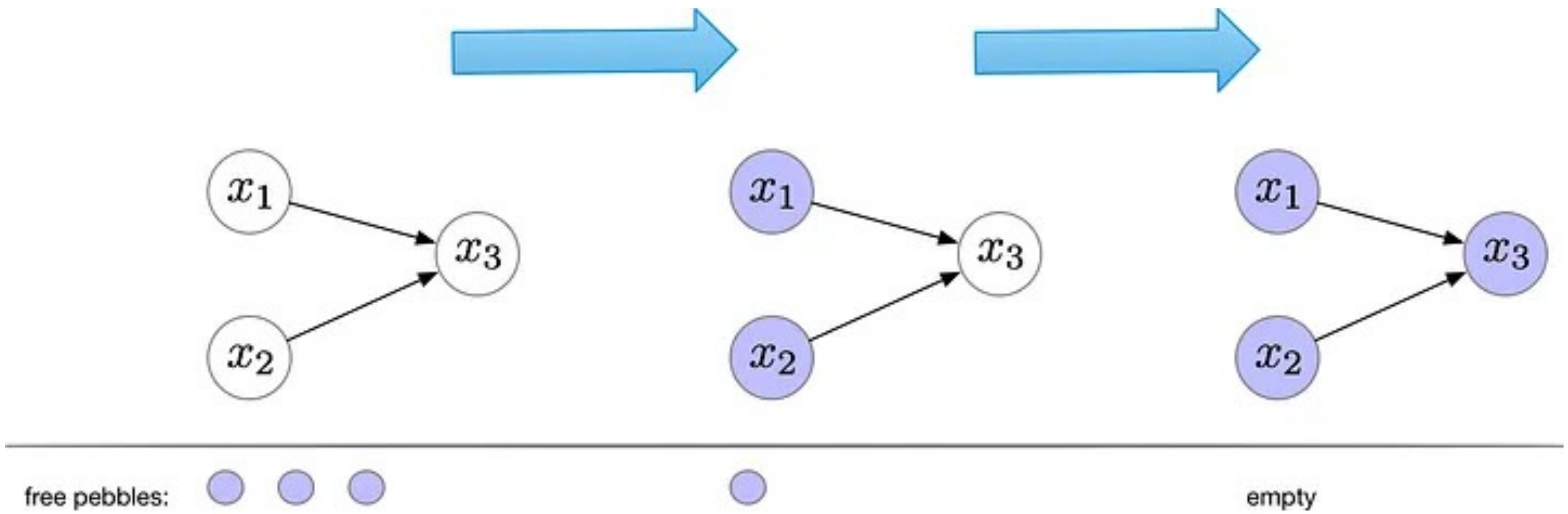


batch size = 1280

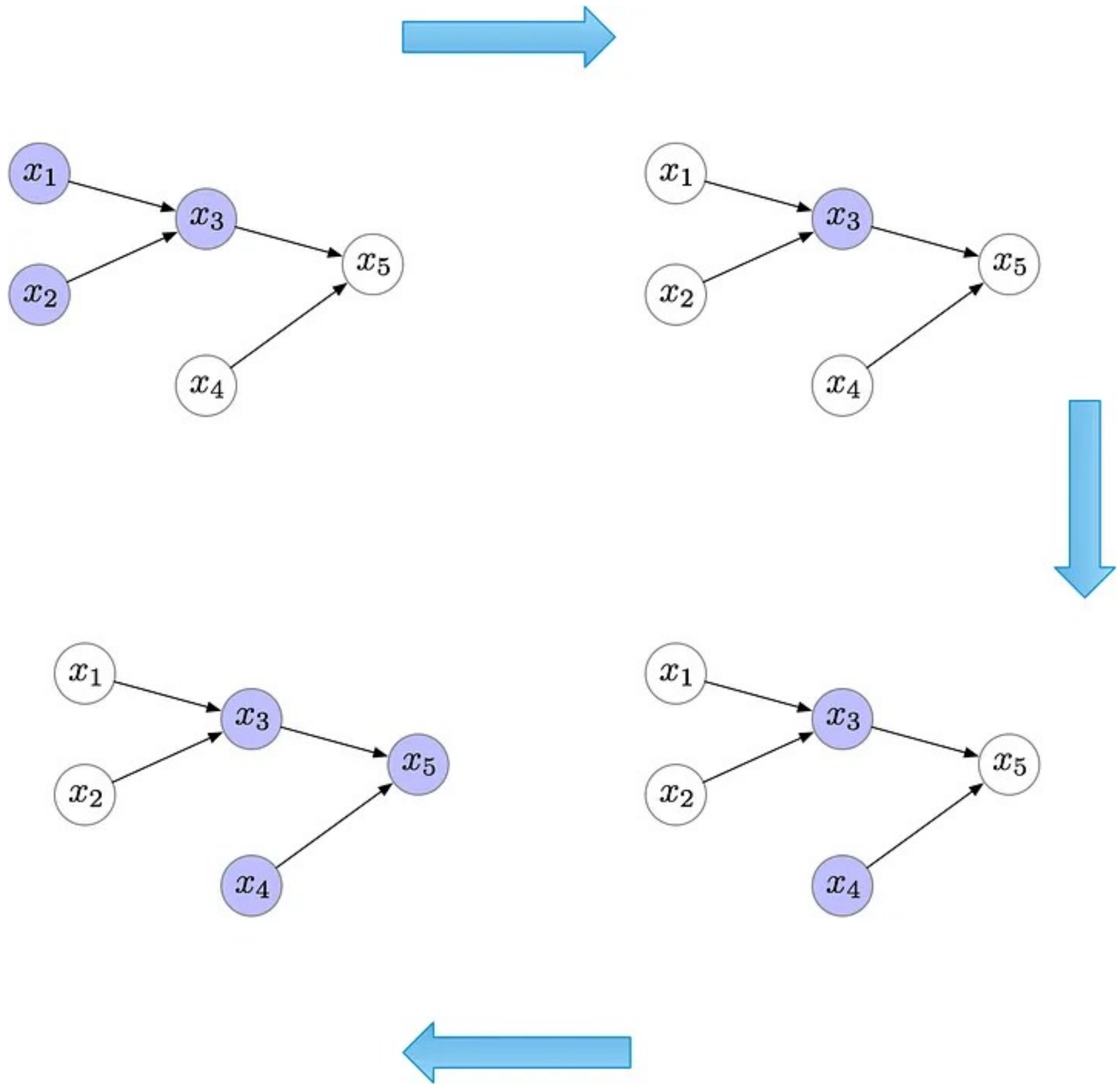


https://medium.com/tensorflow/fitting-larger-networks-into-memory-583e3c758ff9

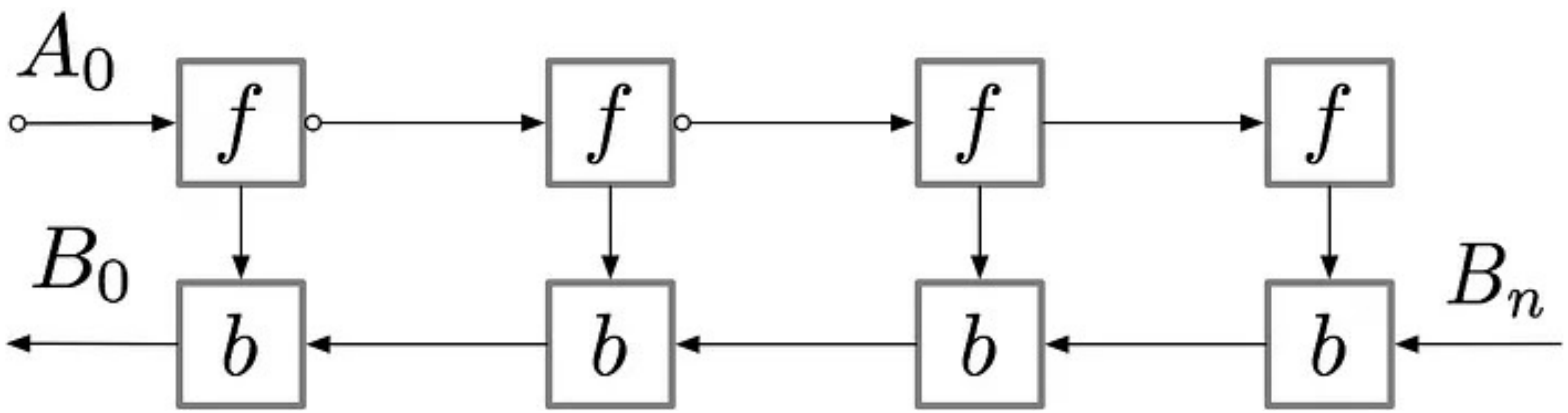
Pebble Analogy



Pebble Analogy



Gradient Computation



Checkpoints every sqrt(n) steps

Memory Requirement	$O(\sqrt{n})$
Compute Requirement	$O(n)$
Forward calcs per node	1 to 2

https://medium.com/tensorflow/fitting-larger-networks-into-memory-583e3c758ff9

FlashAttention-2

Additionally parallelizing the attention computation over sequence length

Partitioning the work between GPU threads to reduce communication and shared memory reads/writes

```
import torch
from transformers import AutoModelForCausalLM, AutoTokenizer, LlamaForCausalLM

model_id = "tiiuae/falcon-7b"
tokenizer = AutoTokenizer.from_pretrained(model_id)

model = AutoModelForCausalLM.from_pretrained(
    model_id,
    torch_dtype=torch.bfloat16,
    attn_implementation="flash_attention_2",
)
```

model’s dtype must be fp16 or bf16

DeepSpeed Good Practices in Training

Version Control and Reproducibility

Use some form of version control

Save the `githash`

Save all the hyperparameters associated with the experiment

Seed your random generators

Specify all the packages and their versions

`requirements.txt` file,
conda `env.yaml` file or
`pyproject.toml` file

DeepSpeed Good Practices in Training

Seed your random generators

```
import torch
torch.manual_seed(0)

import random
random.seed(0)

import numpy as np
np.random.seed(0)
```

https://pytorch.org/docs/stable/notes/randomness.html

DeepSpeed Good Practices in Training

Seed your random generators

CUDA convolution

- Based on parameters, cuDNN will select the fastest algorithm
- Different runs end up with different results

torch.backends.cudnn.benchmark = False

- Causes cuDNN to deterministically select an algorithm

Avoiding nondeterministic algorithms

torch.use_deterministic_algorithms()

https://pytorch.org/docs/stable/notes/randomness.html

DeepSpeed Good Practices in Training

Seed your random generators

DataLoader

Default - each worker gets different random seed

```
def seed_worker(worker_id):
    worker_seed = torch.initial_seed() % 2**32
    numpy.random.seed(worker_seed)
    random.seed(worker_seed)

g = torch.Generator()
g.manual_seed(0)

DataLoader(
    train_dataset,
    batch_size=batch_size,
    num_workers=num_workers,
    worker_init_fn=seed_worker,
    generator=g,
)
```

https://pytorch.org/docs/stable/notes/randomness.html

DeepSpeed Good Practices in Training

Writing Unit Tests

```
def test_model_checkpointing(checkpoint_dir: str):
    train_params = {
        "checkpoint_dir": checkpoint_dir,
        "checkpoint_every": 2,
        "num_layers": 2,
        "num_heads": 4,
        "ff_dim": 64,
        "h_dim": 64,
        "num_iterations": 5,
    }
    exp_dir = train(**train_params)
   # now check that we have 3 checkpoints
    assert len(list(exp_dir.glob("*.pt"))) == 3
    model = create_model(
        num_layers=train_params["num_layers"],
        num_heads=train_params["num_heads"],
        ff_dim=train_params["ff_dim"],
        h_dim=train_params["h_dim"],
        dropout=0.1,
    )
    optimizer = torch.optim.Adam(model.parameters())
    step, model, optimizer = load_model_checkpoint(exp_dir, model, optimizer)
    assert step == 5
    model_state_dict = model.state_dict()
    correct_state_dict = torch.load(exp_dir / "checkpoint.iter_5.pt")
    correct_model_state_dict = correct_state_dict["model"]
```

```
assert set(model_state_dict.keys()) == set(correct_model_state_dict.keys())
    assert all(
        torch.allclose(model_state_dict[key], correct_model_state_dict[key])
        for key in model_state_dict.keys()
    )
    # Finally, try training with the checkpoint
    train_params.pop("checkpoint_dir")
    train_params["load_checkpoint_dir"] = str(exp_dir)
    train_params["num_iterations"] = 10
    train(**train_params)
```