Review for Quiz 2
Topics Covered

- Scaling & Performance
- Data Layouts
- Parallelism
- Sampling and Sketching
- Lying with Statistics & Visualizations
- LLMs
- Cloud Data Systems
Scaling & Performance - General Framework

- 1. Find the bottleneck
  - Print statements, timing, profiler
  - Is it I/O or compute?
- 2. Estimate the gain and cost using Amdahl's law before doing optimizations
- 3. Apply optimizations to the bottleneck
  - Better hardware
  - Better runtimes and algorithms
  - Parallelism
- 4. Iterate the process until satisfied
Performance Metrics

- Latency - response time for processing a request
- Throughput - # requests that can be processed in a unit of time
- Not always inverse of each other due to
  - Queueing
  - Multi-core
  - ...
## Useful Numbers

<table>
<thead>
<tr>
<th>Operation</th>
<th>Time</th>
<th>Throughput</th>
</tr>
</thead>
<tbody>
<tr>
<td>L1 cache reference</td>
<td>0.5 ns</td>
<td></td>
</tr>
<tr>
<td>Branch mispredict</td>
<td>5 ns</td>
<td></td>
</tr>
<tr>
<td>L2 cache reference</td>
<td>7 ns</td>
<td>14x L1 cache</td>
</tr>
<tr>
<td>Mutex lock/unlock</td>
<td>25 ns</td>
<td></td>
</tr>
<tr>
<td>Main memory reference</td>
<td>100 ns</td>
<td>100GB/s, 20x L2 cache, 200x L1 cache</td>
</tr>
<tr>
<td>Compress 1K bytes with Zippy</td>
<td>3,000 ns</td>
<td>3 us</td>
</tr>
<tr>
<td>Send 1K bytes over 1 Gbps network</td>
<td>10,000 ns</td>
<td>10 us</td>
</tr>
<tr>
<td>Read 4K randomly from SSD</td>
<td>150,000 ns</td>
<td>150 us</td>
</tr>
<tr>
<td>Read 1 MB sequentially from memory</td>
<td>250,000 ns</td>
<td>250 us</td>
</tr>
<tr>
<td>Round trip within same datacenter</td>
<td>500,000 ns</td>
<td>500 us</td>
</tr>
<tr>
<td>Read 1 MB sequentially from SSD*</td>
<td>1,000,000 ns</td>
<td>1 ms</td>
</tr>
<tr>
<td>Disk seek</td>
<td>10,000,000 ns</td>
<td>10 ms</td>
</tr>
<tr>
<td>Read 1 MB sequentially from disk</td>
<td>20,000,000 ns</td>
<td>20 ms</td>
</tr>
<tr>
<td>Send packet CA-&gt;Netherlands-&gt;CA</td>
<td>150,000,000 ns</td>
<td>150 ms</td>
</tr>
</tbody>
</table>

**Latency of hardware over the years**
Runtime Improvement

- **Python is slow**
  - Interpreted language
  - Each statement might get interpreted with 10-100x CPU instructions
  - Branch misses and poor processor cache locality

- **Better Python practices for data science applications**
  - Identify hot loops and rewrite them with vectorization library such as numpy/pandas
  - Numpy/pandas internally implement vectorized operations in C
Algorithmic Improvement

- Indexing
  - Clustered tree index vs. linear scan

- Join algorithms complexity
  - Nested loop join: $O(n \times m)$
  - Sort merge join: $O(n \log n + m \log m)$
  - Hash join: $O(n + m)$
Amdahl’s Law: Limit of Performance Improvement

\[ S_{\text{latency}}(s) = \frac{1}{(1 - p) + \frac{p}{s}} \]

\( S_{\text{latency}} \) is the overall speedup in all stages of a task.
\( s \) is the speedup on a stage of the task that we optimize.
\( p \) is the original proportion of time the optimized stage took.
Q1

Suppose you have a data science pipeline with 3 stages, taking 1s, 5s, and 10s. And you come up with a set of optimizations that potentially gain 100x, 10x, and 2x speedup for these three stages correspondingly. If you have one chance, which stage you should optimize first?

A) 1
B) 2
C) 3
Q1

Suppose you have a data science pipeline with 3 stages, taking 1s, 5s, and 10s. And you come up with a set of optimizations that potentially gain 100x, 10x, and 2x speedup for these three stages correspondingly. If you have one chance, which stage you should optimize first?

A) 1  Overall speedup 1.06
B) 2  Overall speedup 1.39
C) 3  Overall speedup 1.45
Q2

For two table S and T where the join columns are indexed with clustered tree index, which of the following join algorithms is likely to have the smallest time and space overhead:

A) Sort Merge Join
B) Nested Loop Join
C) Hash Join
Q2

For two table S and T where the join columns are indexed with clustered tree index, which of the following join algorithms is likely to have the smallest time and space overhead:

A) **Sort Merge Join**  
B) **Nested Loop Join**  
C) **Hash Join**

S,T are already sorted on the join columns, so we can skip sort phase and simply merge using index without using additional space.
Data Layouts

- Data is often multi-dimensional (e.g., tables and ndarray), storage (e.g., memory/disk) is linear
- Data layouts are schemes to linearize data on storage for efficient query processing and maximize locality
- Locality
  - Almost all storage devices favor sequential access
Columns vs. Rows

- **When to use Row format - OLTP**
  - Most of the columns are accessed
  - Frequent modification to individual records
  - Queries access few records

- **When to use Columnar format - OLAP**
  - Only a subset of the columns are accessed
  - High compression ratio is favored
  - Queries access most records
  - Modifications are rare
Combining Row and Columnar formats

- Horizontally partition rows into row groups
- Each group is organized with columnar format
- Allows for skipping row groups while preserving properties of columnar format
- Example: Parquet
Compression

- Entropy coding, e.g., gzip, zlib, ...
  - General purpose, good overall compression

- Delta encoding
  - Encode differences, e.g., 1, 2, 3, 4 -> 1, +1, +1, +1
  - Good for mostly sorted integers/floats

- Bit packing
  - Use fewer bits to encode short integers or categorical data
  - Works well with delta encoding

- Run length encoding
  - Suppress duplicates, e.g., 2, 2, 2, 3, 4, 4, 4, 4, 4, 4 -> 2x3, 3x1, 4x5
  - Good for mostly sorted ints/categorical data with repetition

- Dictionary Encoding
  - Use short code to represent long strings, e.g., AAAAA -> 1, BBBBBB -> 2, C….C -> 3
  - Good for categorical string data with repetitions

- Compressed data are expensive to modify
Z-order

- Spatially close data points are linearized close on storage
- Interleaving bits of multiple dimensions
- \( \text{Zorder}(3,2) \) -
  - \( 3 = 0011 \)
  - \( 2 = 0010 \)
  - \( 0001110 = 14 \)
- \( \text{Zorder}(2,3) \) -
  - \( 3 = 0010 \)
  - \( 1 = 0011 \)
  - \( 0001101 = 13 \)
- Query: find points within a rectangle
Q3

Choose appropriate layouts/techniques for the following application scenarios.

<table>
<thead>
<tr>
<th>Scenario</th>
<th>Layout &amp; Technique</th>
</tr>
</thead>
</table>
| 1) On a massive dataset, compute count of certain string columns with a lot of repeated values | A) Row format  
B) Columnar format  
C) Delta-encoding  
D) Bit-packing  
E) Dictionary Encoding  
F) Z-order |
| 2) Maintaining inventory of a ecommerce website on sales day |  |
| 3) Finding nearby restaurants |  |
Q3

Choose appropriate layouts/techniques for the following application scenarios.

<table>
<thead>
<tr>
<th>Scenario</th>
<th>Layout &amp; Technique</th>
</tr>
</thead>
<tbody>
<tr>
<td>1) On a massive dataset, compute count of certain string columns with a</td>
<td>A)  Row format</td>
</tr>
<tr>
<td>lot of repeated values (BE)</td>
<td>B)  Columnar format</td>
</tr>
<tr>
<td>2) Maintaining inventory of a</td>
<td>C)  Delta-encoding</td>
</tr>
<tr>
<td>ecommerce website on sales day (A)</td>
<td>D)  Bit-packing</td>
</tr>
<tr>
<td>3) Finding nearby restaurants (F)</td>
<td>E)  Dictionary Encoding</td>
</tr>
<tr>
<td></td>
<td>F)  Z-order</td>
</tr>
</tbody>
</table>
Parallelism

- Goal: Make a job faster by running on multiple processors
- Ideally, we want to maintain linear speedup as we throw more hardware at it
- Barriers to linear scaling
  - Start-up times
  - Interference and contention
  - Skews
Python Parallelism

- Threads vs. Processes
- Python GIL: Global Interpreter Lock
  - Work-around: multiprocessing library
Parallel Dataflow

Parallel Dataflow Example

Could send results of filter directly to join instead of buffering

- Directed Acyclic Graph of Operators
  - Data flows from files to output
  - Internally each operator is a parallel job
  - Intermediate results between jobs typically buffered in mem or on disk between tasks
  - May be possible to pipeline directly
Partition Strategies

- **Round Robin/Random Partitioning**
  - No skew
  - Requires repartition for joins

- **Range Partitioning**
  - Allows for predicate pushdown
  - Allows for pruning partitions for joins on partition columns
  - Subject to skew

- **Hash Partitioning**
  - Each partition gets roughly same number of records
  - Allows for equality predicate pushdown
  - Subject to skew on repeated values
Shuffle Join

- Joining on columns that are not partition columns
- Shuffle join
  - Repartition the data on the join key
  - Perform local join normally
Q4

- Suppose you have table T with 9M records and S with 1M records. Let’s say T/S are randomly partitioned/hash partitioned over a cluster of 3 nodes with no skew. The network can transfer 1M records per second in aggregate. How much time for each of the following join strategies would spend on network transfer?
- A) Broadcasts S across 3 nodes and perform join locally with T
- B) Repartition T across 3 nodes according to S’s hash partitioning and perform join locally with S
Q4

- Suppose you have table T with 9M records and S with 1M records. Let’s say T/S are randomly partitioned/hash partitioned over a cluster of 3 nodes with no skew. The network can transfer 1M records per second in aggregate. How much time for each of the following join strategies would spend on network transfer?
- A) Broadcasts S across 3 nodes and perform join locally with T  \[3 \times \frac{2M}{1M/s} = 6s\]
- B) Repartition T across 3 nodes according to S’s hash partitioning and perform join locally with S  \[3 \times 2M / 1M/s = 6s\]
Spark

- High-level distributed computing framework
  - Users do not have control over location of compute and storage
- Core abstractions
  - RDD and DataFrame
- Lazy-evaluation on parallel DAG
BREAK
Ray

- Low-level distributed computing framework
  - Provides user w/more control over location of compute and data
  - Support for stateful execution (Actors) and stateless execution (Tasks) in Ray Core
  - Ray Core forms the basis of Ray’s ML libraries

- Core abstractions
  - Actor and tasks
  - Futures

- Eager-evaluation
The Ray API

Tasks

```python
@ray.remote
def f(shape):
    return np.zeros(shape)

@ray.remote
def add(a, b):
    return a + b
```

```python
o1 = f.remote([5, 5])
o2 = f.remote([5, 5])
o3 = add.remote(o1, o2)
result = ray.get(o3)
```

o1 is a:

...`future`: The eventual value will be computed by `f`.

...`remote reference`: The value may be stored on a remote node (in Ray’s distributed object store).
## The Ray API

### Tasks

```python
@ray.remote
def f(shape):
    return np.zeros(shape)

@ray.remote
def add(a, b):
    return a + b
```

```python
o1 = f.remote([5, 5])
o2 = f.remote([5, 5])
o3 = add.remote(o1, o2)
result = ray.get(o3)
```

### Actors

```python
@ray.remote
class Counter(object):
    def __init__(self):
        self.value = 0
    def inc(self):
        self.value += 1
        return self.value
```

```python
c = Counter.remote()
o4 = c.inc.remote()
o5 = c.inc.remote()
# Returns [1, 2].
result = ray.get([o4, o5])
```
Ray API

Other details:

- `ray.get(refs)` is a blocking operation which returns the true computed value of the futures `refs`.

- `ray.wait(refs, num_returns)` will block until `num_returns` futures in `refs` have finished, and then return a tuple with the finished futures and unfinished futures.
Q5

Assume you have a Ray cluster with 4 CPUs, no scheduling overhead, and no network latency. Assume any race condition prints happen in the order in which tasks are submitted. What output would be printed by the following program?

```python
@ray.remote
def echo(idx, sleep_seconds):
    print(f"START IDX: {idx}"
    time.sleep(sleep_seconds)
    print(f"END IDX: {idx}"
    return f"echo-{idx}"

refs = [echo.remote(idx, 10) for idx in [1, 2, 3]]
moreRefs = [echo.remote(idx, idx) for idx in [4, 5, 6]]
allRefs = refs + moreRefs
echos = ray.get(allRefs)
```
Q5

Assume you have a Ray cluster with 4 CPUs, no scheduling overhead, and no network latency. Assume any race condition prints happen in the order in which tasks are submitted. What output would be printed by the following program?

```python
@ray.remote
def echo(idx, sleep_seconds):
    print(f"START IDX: {idx}""
    time.sleep(sleep_seconds)
    print(f"END IDX: {idx}""
    return f"echo-{idx}"

refs = [echo.remote(idx, 10) for idx in [1, 2, 3]]
more.refs = [echo.remote(idx, idx) for idx in [4, 5, 6]]
all.refs = refs + more.refs
echos = ray.get(all.refs)
```

Answer:

START IDX: 1
START IDX: 2
START IDX: 3
START IDX: 4
END IDX: 4
START IDX: 5
END IDX: 5
START IDX: 6
END IDX: 1
END IDX: 2
END IDX: 3
Sampling & Sketching Overview

- Random Sampling to Estimate Statistics
- The Bootstrap Method
- HyperLogLog
- CountMin
Random Sampling to Est. Statistics

- For very large datasets, it can be time consuming to compute some statistic (e.g. the mean, sum, std. dev., etc.) over the entire dataset
  - This is especially annoying for use cases where we don’t need a precise number

- **Idea:** let’s save order(s) of magnitude of computation by computing an estimate of the statistic using a sample of the dataset
  - Typically you also want to compute some error bounds / confidence interval as well

- Thanks to the **Central Limit Theorem**, for estimates of some simple statistics like mean, sum, and count, we can compute closed form estimates + bounds

- For other statistics (e.g. median of a dataset, or mean of some ML model output) the bootstrap is required
Bootstrap Method

Given a function $F$ and a sample $S$ of size $N$, with parameter $K$ (the number of bootstraps)

Goal is +/- $p$ confidence interval

For $i$ in $1 .. K$
  - $S_{\text{new}} = \text{sample of size } N \text{ of } S \text{ with replacement}$
  - $\text{Results}[i] = F(S_{\text{new}})$

Sort results, return $p$, $1-p$ percentile of results
Example

Data:
[36, 23, 7, 25, 27, 31, 27, 10, 11, 8, 21, 4, 41, 0, 20, 5, 0, 36, 40, 10, 12, 31, 24, 2, 28, 8, 9, 25, 48, 43, 40, 2, 26, 0, 2, 5, 32, 9, 0, 10, 33, 1, 23, 7, 39, 18, 32, 16, 40, 0, 42, 28, 28, 26, 42, 0, 45, 25, 10, 13, 31, 3, 11, 28, 25, 31, 2, 6, 34, 19, 48, 27, 48, 39, 40, 6, 3, 28, 26, 19, 34, 38, 42, 1, 47, 22, 7, 36, 38, 35, 35, 42, 49, 41, 40, 11, 10, 1, 1]

Sample:
[25, 10, 35, 25, 23, 0, 20, 24, 22, 25, 6, 42, 40, 38, 40, 4, 8, 16, 38, 8]

Resample 1: [42, 40, 8, 25, 0, 42, 24, 0, 16, 42, 23, 25, 25, 10, 40]  Mean = 24.1
Resample 2: [23, 25, 10, 42, 23, 0, 0, 24, 23, 23, 38, 25, 16, 35, 25]  Mean = 22.1
Resample 3: [6, 38, 40, 23, 23, 40, 23, 4, 8, 25, 4, 8, 25, 20, 0]  Mean = 19.13
## Resulting Means after 100 runs


Confidence interval of mean 16.6 ... 28.87

Q: what % confidence interval is this?

Confidence interval of mean 16.6 \ldots 28.87

Q: what % confidence interval is this?
A: 90%
Sampling vs. Sketching

- Sampling works well for descriptive statistics of entire distributions, as (large enough) samples provide a good approx. of the underlying dataset.

- Sampling does not work well for “extreme” statistics (e.g. max, min, # of distinct values).

- Sketching algorithms: approximate probabilistic algos for est. these stats.
  - # distinct values → HyperLogLog
  - Top-K values → CountMin
  - And others
HyperLogLog v0

- Recall key insight: in expectation, you will need $2^k$ samples in order to observe a value with $k$ leading 0s
- HyperLogLog v0 algorithm (the basic idea):

1. Given a vector of values $V$, hash each element $v$ to some large integer $H(v)$
2. $\text{maxZeros} = 0$
3. For each $h$ in $H(V)$:
   a. $\text{zeros} =$ # of leading zeros in $h$
   b. $\text{maxZeros} = \max(\text{zeros}, \text{maxZeros})$
4. $\text{distinctVals} \sim 2^{\text{maxZeros}}$

- This is an accurate (i.e. unbiased) but noisy estimator, can be made less noisy by averaging multiple estimators
Hyperloglog Algorithm – Approach 1

Idea: split hash value into $m$ “bucket” bits and $128 - m$ “value” bits; store $2^m$ max’s

| m “bucket” bits | 128 – m hash bits |

Given a vector of values, $V$, compute $H(v)$ for all $v$ in $V$

$H$ is a hash function that goes from $v$ to a large random integer

MaxZeros = [0, 0, ..] // length $2^m$

For each $h$ in $H(v)$ $\forall v$ in $V$:

- bucket = bits 0 ... $m-1$ of $h$
- value = bits $m$ ... 128 of $h$
- zeros = count the number of leading zeros in value
- $\text{MaxZeros}[\text{bucket}] = \max(\text{zeros}, \text{MaxZeros}[\text{bucket}])$

Distinct vals = $\text{avg}(2^{\text{MaxZeros}[0]}, ..., 2^{\text{MaxZeros}[2^m]})$
CountMin

- Setup: suppose we have two data streams:
  - UserIds of user’s visiting website
  - OS version of user’s phone/laptop

- Goal: estimate most frequent user(s), most frequent OS(es)
  - These aren’t the only statistics we could estimate, but they are good examples

- Bad idea: store entire history of website visit data (grows unbounded)
CountMin

- **Simple Key Idea:** keep a fixed-size table of $N$ counters, hash incoming item(s) to location in table, and increment that counter
  - To lookup the count for an item, simply hash item and read counter value in its table loc.
  - Can only over-estimate true count

- **Improvement:** Keep an $(M, N)$ table of counters, use $M$ hash functions to hash each incoming item to $M$ locations, increment each of the $M$ counters
  - To lookup the count for an item, hash the item to all $M$ locations and read the min of the counters
  - This can only overestimate if all $M$ hash fcns collide
Lying w/Statistics and Visualizations Overview

- Problems w/summary stats
- P-values
- Simpson’s Paradox
Problems w/Summary Stats

- Summary statistics disguise the true distribution of the underlying data
  - E.g. “Michael Jordan and I have combined to win 6 NBA Championships”

- Terms are often un(der)defined
  - E.g. “Average” is used to mean mean, median, or sometimes even mode
  - E.g. “MIT Graduates” can refer to all undergrads, all undergrads + grad students, or some random subset the statmaker had in their dataset

- Surveys are likely to have sample bias
P-values

- When conducting a scientific study we often want to perform a statistical test to compare our findings to the null hypothesis.

- For example, I hypothesize that spending time outside increases students’ reported well-being.

- The null hypothesis: spending time outside does not increase students’ reported well-being.
The *p value* (Type I error) is the probability of obtaining an effect equal to or more extreme than the one observed presuming the null hypothesis of no effect is true.
P-values

Q: what does it mean for a study to have a p-value < 0.05?

A) The chances are greater than 1 in 20 that a difference would be found if the study were repeated.

B) The probability is less than 1 in 20 that a difference this large or larger could occur by chance alone.

C) The probability is greater than 1 in 20 that a difference this large or larger could occur by chance alone.

D) The chance is 95% that the study is correct

E) None of the above
Simpson’s Paradox

- **Ex:** assume 100 cancer patients either have small or large tumors
  - (large is worse for life expectancy, let’s say we have 50 patients w/small and 50 w/large)
- Suppose we split the patients into test two populations (50 in each)
- Population1 gets Treatment A and population2 gets Treatment B
- After running the study, we find that:
  - 40/50 patients in the population receiving treatment B survive
  - 30/50 patients in the population receiving treatment A survive
  - Patients w/small tumors are more likely to survive if they take treatment A
  - Patients w/large tumors are more likely to survive if they take treatment A
- **Q:** How is this possible?

Nice visual explainer on Simpson’s Paradox (scroll down to “Aside: Simpson’s Paradox”)
Simpson’s Paradox [see lec. slides for formalism]

- **Ex:** assume 100 cancer patients either have small or large tumors
  - (large is worse for life expectancy, let’s say we have 50 patients w/small and 50 w/large)
- Suppose we split the patients into test two populations (50 in each)
- Population1 gets Treatment A and population2 gets Treatment B
- After running the study, we find that:
  - 30/50 patients in the population receiving treatment A survive
  - 40/50 patients in the population receiving treatment B survive
  - Patients w/small tumors are more likely to survive if they take treatment A
  - Patients w/large tumors are more likely to survive if they take treatment A
  - **Q:** How is this possible? Short answer: Pop1 sample skewed towards large tumors

**A:** suppose Pop1 had 40 + 10 patients with (large + small) tumors and pop2 had 10 + 40. Suppose 22 patients w/large tumors and 8 patients w/small tumors survive in pop1 and 5 + 35 survive in pop2; Then 21/40 and 9/10 survive w/A and 5/10 and 35/40 survive w/B.
LLMs Overview

- Sequence Models
- Perplexity
- Attention
- Transformers
Sequence Models

- At its core, a sequence model takes a sequence of tokens as input and produces a sequence of tokens as output.

- We can build the simplest sequence model if we have an underlying model which can compute $P(\text{next\_token} | \text{previous\_token(s)})$.
  - $P(\text{sentence}) = P(w_1) \times P(w_2 | w_1) \times P(w_3 | w_2, w_1) \times \ldots \times P(w_n | w_{n-1}, \ldots, w_1)$

- **Idea #1 (k-gram model):** take a big dataset and compute $P(w_k | w_{\{k-1\}}, \ldots, w_1)$ as the count of occurrences of $(w_1, \ldots, w_k)$ divided by the number of occurrences of $(w_1, \ldots, w_{\{k-1\}})$.
  - Intuitively, this measures “what pct. of the time that $(w_1, \ldots, w_{\{k-1\}})$ did it finish with $w_k$?”
Perplexity

- How do we tell if a sequence model is “good”?

- **Idea:** take an unseen test corpus (set of sentences) and measure the inverse probability of generating those sentences with your sequence model
  - If your model is good, and test corpus sentences are “real”, then probability should be high, so inverse probability (perplexity) should be low

\[
PP(W) = P(w_1 \ldots w_N)^{-\frac{1}{N}}
\]

- Pros: easy to compute, generalizes across tasks, works for any language
- Cons: only works for models which output probabilities, can’t compare models w/diff. vocabularies, notion of a “good” perplexity value is vague
Going Beyond Fixed-Context Sequence Models

- K-gram model has fixed context
- Recurrent neural network (RNN) models summarizes entire history of sequence in its hidden state

- Can use this as building block for encoder/decoder architecture
  - Encoder takes in input seq. and computes hidden state
  - Decoder takes hidden state and produces output seq.
    - (Both trained to predict next token)

- **Problem:** Naive encoder-decoders suffer from encoder bottleneck (compressing all info into single context vector) and vanishing gradients
Attention

- **Key Idea:** Have decoder learn to attend to specific context vectors (produced by encoder) when decoding output at some position in sequence
  - (For implementation details, I recommend reading the blog post linked in our slides): [LINK]

Bahdanau Attention

• Karpathy, AI’s bard: “The context vector from encoder is a weighted sum of hidden states of words of the encoding”
• Those attention weights are themselves computed by looking at current decoder state and encoder values

Bahdanau, Cho, Bengio, “Neural Machine Translation By Jointly Learning to Align and Translate”
Attention

- **Self-Attention**: At a very high-level — each vector becomes a weighted avg. of its neighbors and itself
- Key, query, and value computed for each input token vector
- Output (A2) is computed as a weighted average of values weighted by their attention weights
Transformers

- Don’t need to know architecture specifics
- Key ideas:
  - Uses multi-head attention to compute self-attention mechanism multiple times in parallel
    - Each head can pay attention to different aspects of input
  - Is computationally more efficient than traditional Enc-Dec b/c entire input and output sequence(s) can be fed into transformer at once during training
    - (Is still autoregressive for generation)
Cloud Data Systems Overview

- How is the cloud different
- Disaggregated storage challenges & advantages
- Performance Optimizations
How is the Cloud Different?

- (Borderline infinite) elastic scaling of compute and storage
  - Rather than provisioning your own on-prem resources for max utilization / worst case scenario, you can pay for what you need, when you need it

- Durable networked storage layer (e.g. S3 in AWS)
  - Highly available and highly durable (would take global armageddon to lose your data)
  - Offers ~100s of MB/s bandwidth (okayish)
  - Comes w/low latency (hundreds of ms / request)
Disaggregated Storage Challenges & Advantages

Challenges:
- Shared disk can scale worse than shared nothing for certain queries

Advantages:
- Can scale compute up/down to meet demands of workload on a per-query basis
- Shared disk means data lives in one place (if node dies, other nodes can get/repartition data by reading from S3)
- Shared disk w/S3 gives great durability guarantees
- Can paper over low latency reads from S3 w/caching
Performance Optimizations [see slides for specifics]

- Tiered Data Storage Layer
  - Avoids going to S3 on the critical path much of the time

- Table Clustering + Block Skipping
  - Allows system to skip reading entire “blocks” (i.e. partitions) of data by looking at metadata for per-block key ranges

- Consistent Hashing for data affinity + caching
  - Minimizes transfer of partitions between worker nodes when new nodes join / leave system

- Columnar Data Layout, Vectorized Execution, and more