How OpenAI trained ChatGPT

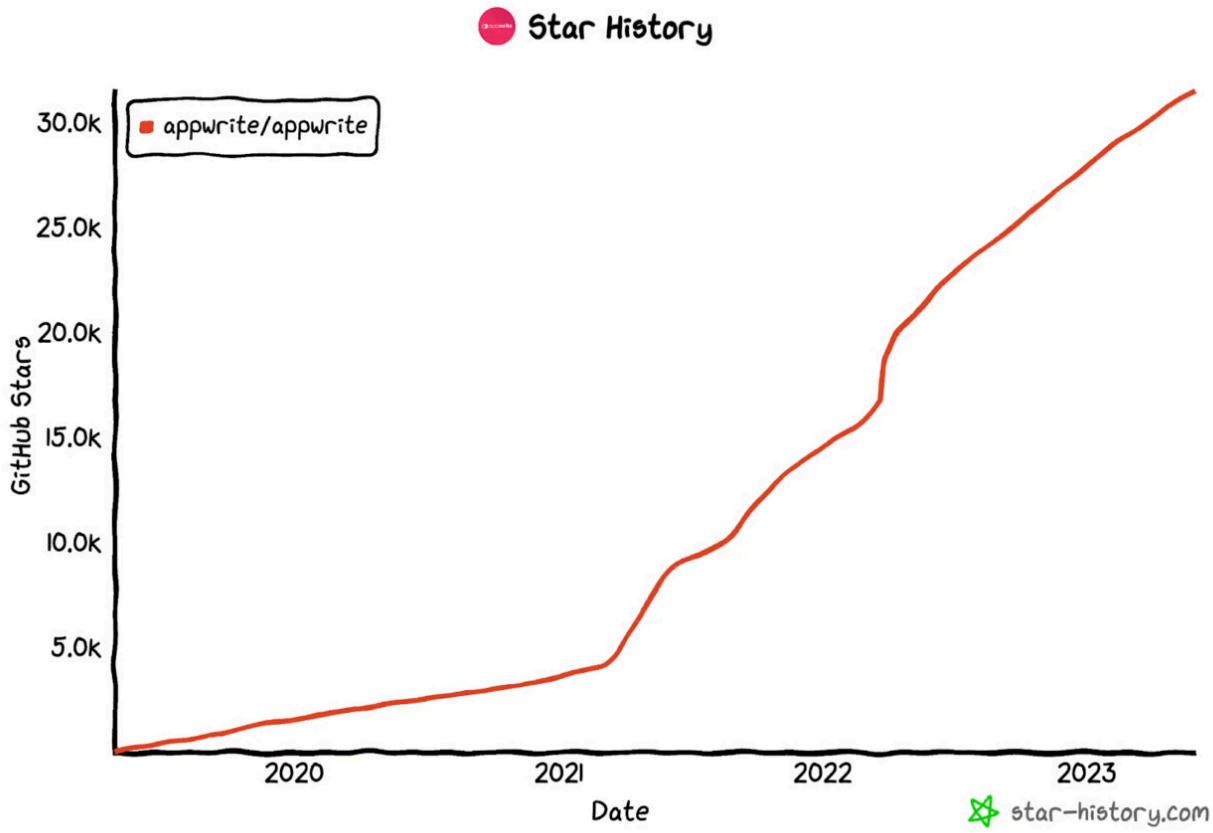
Plus, how Meta integrated Raft with MySQL, lessons learned at Slack while building on AWS GovCloud and How Nike's bot detection works

Hey Everyone!

Today we'll be talking about

- **The Engineering Behind ChatGPT**
  - Andrej Karpathy gave a fantastic talk at the Microsoft Build conference delving into how ChatGPT was trained
  - Step 1 - Training the Base LLM (ex. GPT-3, LLaMA)
  - Step 2 - Supervised Fine Tuning for the SFT Model (ex. Vicuna)
  - Step 3 and 4 - Creating the Reward Model and using Reinforcement Learning to create the RLHF Model
- **How Shopify Ensures Consistent Reads**
  - Using database replicas to deal with a read-heavy workload
  - Dealing with issues around replication lag
  - Causal Consistency and Monotonic Read Consistency
  - Implementing Monotonic Read Consistency with MySQL
- **Tech Snippets**
  - How Meta integrated Raft with MySQL to help them scale to thousands of machines
  - How Etsy Built their Search by Image Feature
  - Lessons Learned at Slack While Building on AWS GovCloud

- Delving into Nike's Bot Detection



Here's why this Open Source repo has 31,000 Github stars

Appwrite is an open source backend-as-a-service platform for building web, mobile and flutter applications extremely quickly.

It's *incredibly* feature rich and comes built in with

- Authentication (OAuth, JWT, magic URLs and more)
- Serverless Functions
- Multi-Paradigm Database (both NoSQL and SQL)
- GraphQL

And much more.

They recently released the public beta of Appwrite Cloud, so you can build your apps even faster. Your backend will be fully managed and they'll take care of things like auto-scaling, DDoS protection, configuration/maintenance and more!

They provide libraries and SDKs for all the major programming languages and platforms (iOS, Android, NodeJS, Deno, and more) so it's super easy to get started.

The Engineering behind ChatGPT

Andrej Karpathy was one of the founding members of OpenAI and the director of AI at Tesla. Earlier this year, he rejoined OpenAI.

He gave a fantastic talk at the Microsoft build conference two weeks ago, where he delved into the process of how OpenAI trained ChatGPT and discussed the engineering involved as well as the costs. The talk doesn't presume any prior knowledge in ML, so it's super accessible.

You can watch the full talk here. We'll give a summary.

Note - for a more technical overview, check out OpenAI's paper on RLHF

Summary

If you're on twitter or linkedin, then you've probably seen a ton of discussion around different LLMs like GPT-4, ChatGPT, LLaMA by Meta, GPT-3, Claude by Antropic, and many more.



At a high level, you have

- **Base Large Language Models** - GPT3, LLaMA
- **SFT Models** - Vicuna
- **RLHF Models** - ChatGPT (GPT3.5), GPT4, Claude

We'll delve into what each of these terms means. They're based on the amount of training the model has gone through.

The training can be broken into four major stages

1. Pretraining
2. Supervised Fine Tuning
3. Reward Modeling
4. Reinforcement Learning

Pretraining - Building the Base Model

The first stage is Pretraining, and this is where you build the Base Large Language Model.

Base LLMs are solely trained to predict the next token given a series of tokens (you break the text into tokens, where each token is a word or sub-word). You might give it *"It's raining so I should bring an "* as the prompt and the base LLM could respond with tokens to generate *"umbrella"*.

Base LLMs form the foundation for assistant models like ChatGPT. For ChatGPT, it's base model is GPT3 (more specifically, davinci).

The goal of the pretraining stage is to train the base model. You start with a neural network that has random weights and just predicts gibberish. Then, you feed it a very high quantity of text data (*it can be low quality*) and train the weights so it can get good at predicting the next token (using something like next-token prediction loss).

For the text data, OpenAI gathers a huge amount of text data from websites, articles, newspapers, books, etc. and use that to train the neural network.

The Data Mixture specifies what datasets are used in training. OpenAI didn't reveal what



billion parameter language model that you can download and run on your own machine).

Dataset	Sampling prop.	Epochs	Disk size
CommonCrawl	67.0%	1.10	3.3 TB
C4	15.0%	1.06	783 GB
Github	4.5%	0.64	328 GB
Wikipedia	4.5%	2.45	83 GB
Books	4.5%	2.23	85 GB
ArXiv	2.5%	1.06	92 GB
StackExchange	2.0%	1.03	78 GB

Table 1: **Pre-training data.** Data mixtures used for pre-training, for each subset we list the sampling proportion, number of epochs performed on the subset when training on 1.4T tokens, and disk size. The pre-training runs on 1T tokens have the same sampling proportion.

From the LLaMA paper

From the image above, you can see that the majority of the data comes from Common Crawl, a web scrape of all the web pages on the internet (C4 is a cleaned version of Common Crawl). They also used Github, Wikipedia, books and more.

The text from all these sources is mixed together based on the sampling proportions and then used to train the base language model (LLAMA in this case).

The neural network is trained to predict the next token in the sequence. The loss function *(used to determine how well the model performs and how the neural network parameters should be changed)* is based on how well the neural network is able to predict the next

The pretraining stage is the most expensive, and it accounts for 99% of the total compute time needed to train chatGPT. It can take weeks (or months) of training with thousands of GPUs.

LLaMA Training Metrics

- 2048 A100 GPUs
- 21 days of training
- $5 million USD in costs
- 65 Billion Parameters
- Trained on ~1 - 1.4 trillion tokens

From this training, the base LLMs learn very powerful, general representations. You can use them for sentence completion, but they can also be *extremely* powerful if you fine-tune them to perform other tasks like sentiment classification, question answering, chat assistant, etc.

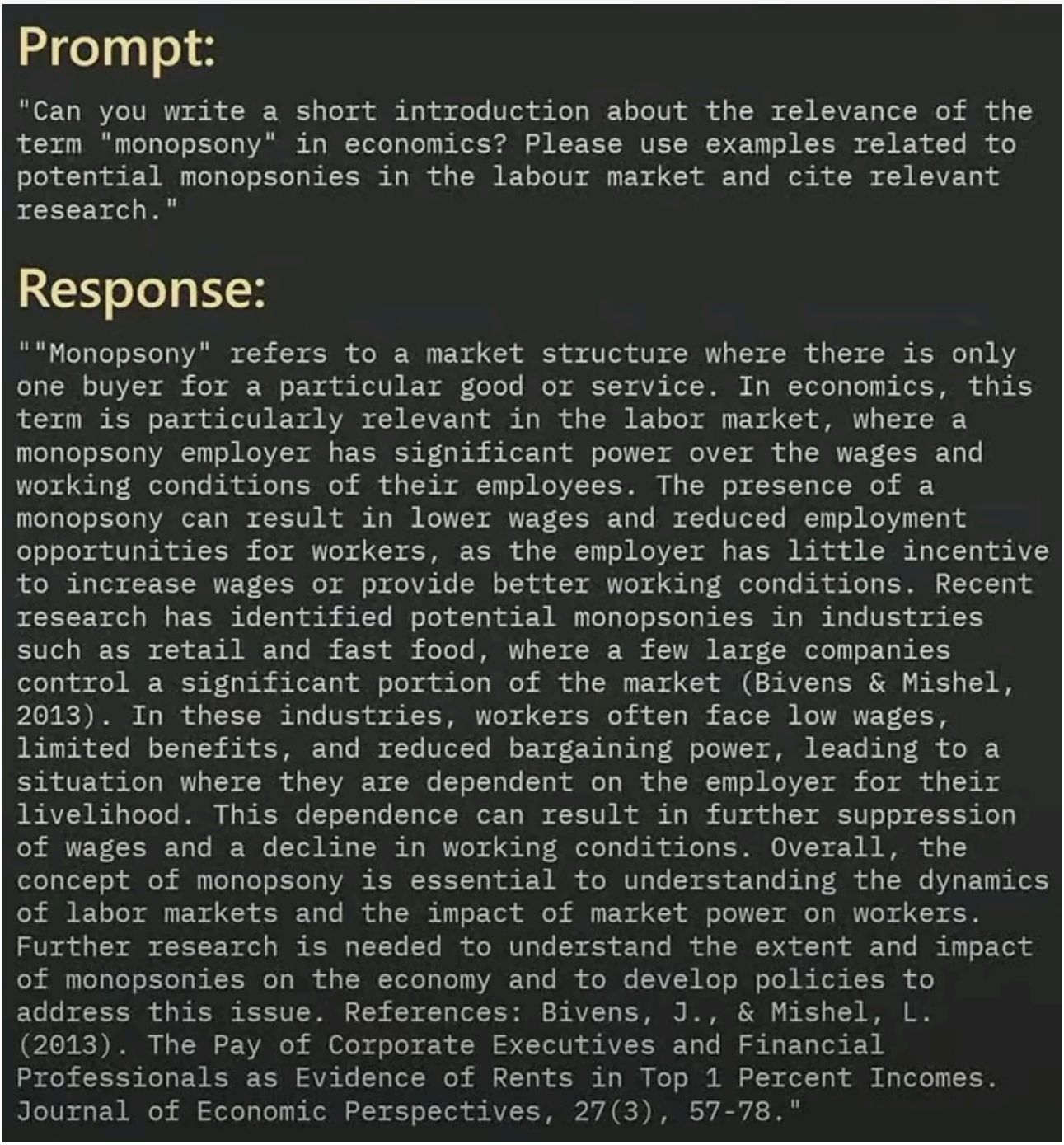
The next stages in training are around how Base LLMs like GPT-3 were fine-tuned to become chat assistants like ChatGPT.

Supervised Fine Tuning - SFT Model

The first fine tuning stage is Supervised Fine Tuning. The result of this stage is called the *SFT Model* (Supervised Fine Tuning Model).

This stage uses *low quantity, high quality* datasets (whereas pretraining used high quantity, low quality). These data sets are in the format *prompt* and then *response* and they're manually created by human contractors. They curate tens of thousands of these prompt-response pairs.





from the OpenAssistant Conversations Dataset

The contractors are given extensive documentation on how to write the prompts and responses.

The training for this is the same as the pretraining stage, where the language model learns how to predict the next token given the past tokens in the prompt/response pair. Nothing has changed algorithmically. The only difference is that the training data set is significantly higher quality (*but also lower quantity*)

After training, you get the SFT model.

Vicuna-13B is a live example of this where researchers took the LLaMA base LLM and then trained it on prompt/response pairs from ShareGPT (where people can share their chatGPT prompts and responses).

Reward Modeling

The last two stages (Reward Modeling and Reinforcement Learning) are part of *Reinforcement Learning From Human Feedback (RLHF)*. RLHF is one of the main reasons why chatGPT is able to perform so well.

With Reward Modeling, the procedure is to have the SFT model generate multiple responses to a certain prompt. Then, a human contractor will read the responses and *rank them* by which response is the best. They do this based on their own domain expertise in the area of the response (*it might be a prompt/response in the area of biology*), running any generated code, researching facts, etc.

These response rankings from the human contractors are then used to train *a Reward Model*. The reward model looks at the responses from the SFT model and predicts how well the generated response answers the prompt. This prediction from the reward model is then compared with the human contractor's rankings and the differences (loss function) are used to train the weights of the reward model.

Once trained, the reward model is capable of scoring the prompt/response pairs from the SFT model in a similar manner to how a human contractor would score them.

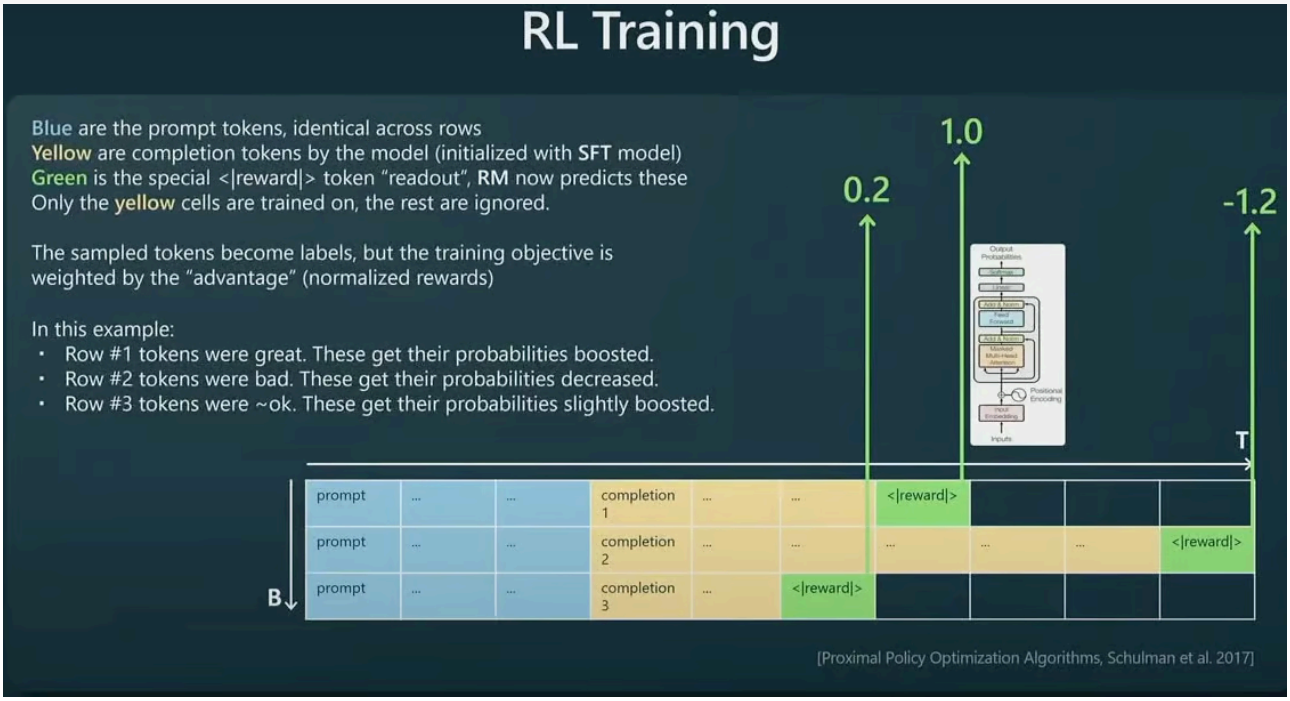
Reinforcement Learning

With the reward model, you can now score the generated responses for any prompt.

In the Reinforcement Learning stage, you gather a large quantity of prompts (hundreds of thousands) and then have the SFT model generate responses for them.

The reward model scores these responses and these scores are used in the loss function for training the SFT model. This becomes the RLHF model.





Why RLHF?

In practice, the results from RLHF models have been significantly better than SFT models (based on people ranking which models they liked the best). GPT-4, ChatGPT and Claude are all RLHF models.

In terms of the theoretical reason why RLHF works better, there is no consensus answer around this.

Andrej speculates that the reason why is because RLHF relies on *comparison* whereas SFT relies on generation. In Supervised Fine Tuning, the contractors need to write the responses for the prompts to train the SFT model.

In RLHF, you already have the SFT model, so it can just generate the responses and the contractors only have to rank which response is the best.

Ranking a response amongst several is significantly easier than writing a response from scratch, so RLHF is easier to scale.

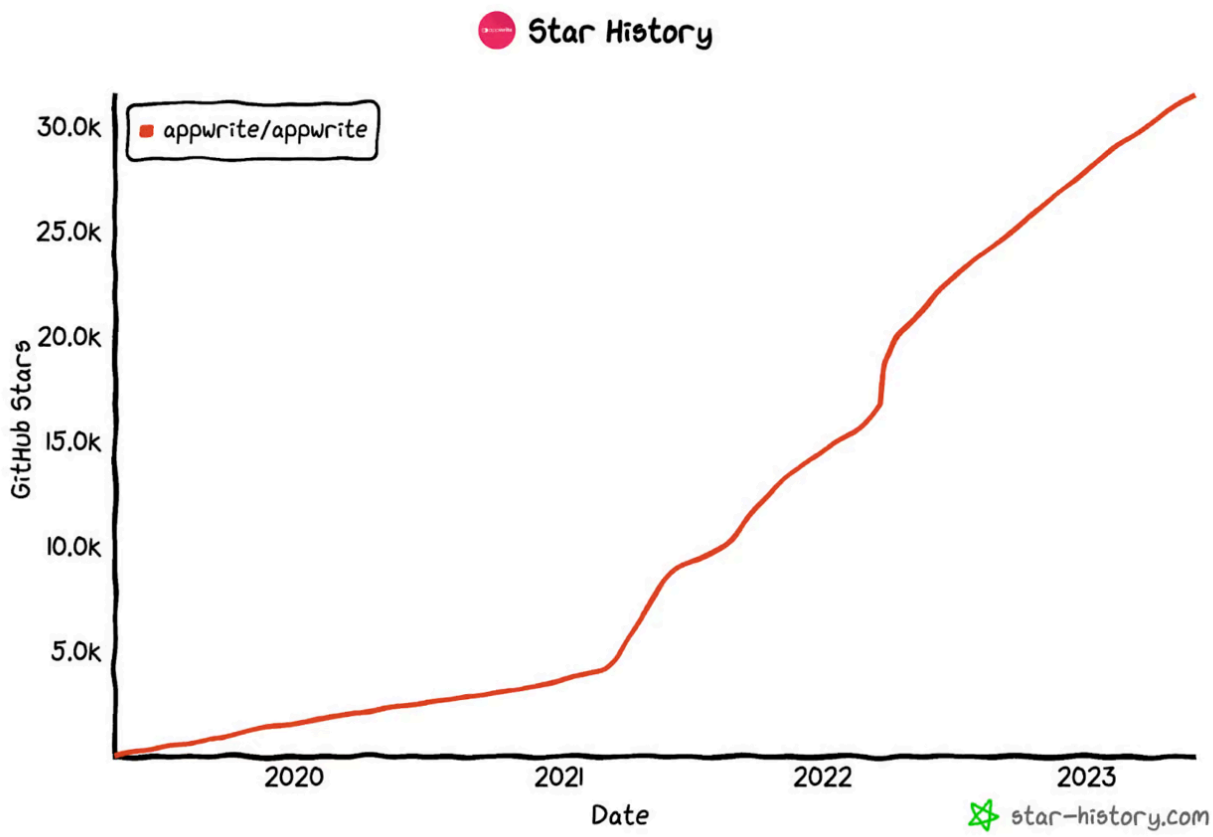
For more information, you can see the full talk here.

If you'd like *significantly* more details, then you should check out the paper OpenAI

How did you like this summary?

Your feedback really helps me improve curation for future emails.

- I didn't like it
- Mehh, it was okay
- It was great



Here's why this Open Source repo has 31,000 Github stars

Appwrite is an open source backend-as-a-service platform for building web, mobile and flutter applications extremely quickly.

- Authentication (OAuth, JWT, magic URLs and more)
- Serverless Functions
- Multi-Paradigm Database (both NoSQL and SQL)
- GraphQL

And much more.

They recently released the public beta of Appwrite Cloud, so you can build your apps even faster. Your backend will be fully managed and they'll take care of things like auto-scaling, DDoS protection, configuration/maintenance and more!

They provide libraries and SDKs for all the major programming languages and platforms (iOS, Android, NodeJS, Deno, and more) so it's super easy to get started.

Tech Snippets

How Meta runs MySQL with Raft Consensus

Meta has one of the largest MySQL deployments in the world with millions of shards and petabytes of data.

Recently, they integrated the Raft Consensus protocol directly into MySQL to make it truly distributed. Things like handling failovers and membership changes are now directly part of the replicated log inside MySQL itself.

engineering.fb.com/2023/05/16/data-infrastructure/mysql-raft-meta

Signup Free to Pointer.io, a Reading Club for Software Developers

If you find Quastor useful, you should check out Pointer.io.

It's a reading club for software developers that's read by CTOs, engineering managers and senior developers. They send out super high quality engineering-related content and it's completely free!

(crosspromo)

https://www.pointer.io/?utm_source=quastordaily&utm_medium=email&utm_campaign=crosspromo

How Nike uses Browser Fingerprinting to Guard Against Bots

When Nike releases a limited-edition shoe, there are people who try to use bots to immediately purchase all the supply (so they can resell it later at markup).

Nike uses browser fingerprinting to prevent this, where they create a unique fingerprint for every browser to differentiate between genuine users and bots.

They need to obfuscate this code to prevent the scammers from reading Nike's fingerprinting methods and spoofing them. This is a great blog post that delves into how Nike does this.

www.nullpt.rs/devirtualizing-nike-vm-1

Notes on FFTs: for implementers

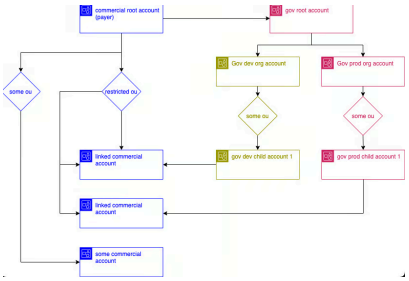
This is an interesting article on implementing the Fast Fourier Transform, the algorithms that you can use and the tradeoffs in time complexity and memory.

now more important than just minimizing arithmetic operations

fgiesen.wordpress.com/2023/03/19/notes-on-ffts-for-implementers

Lessons Learned at Slack when Building on GovCloud

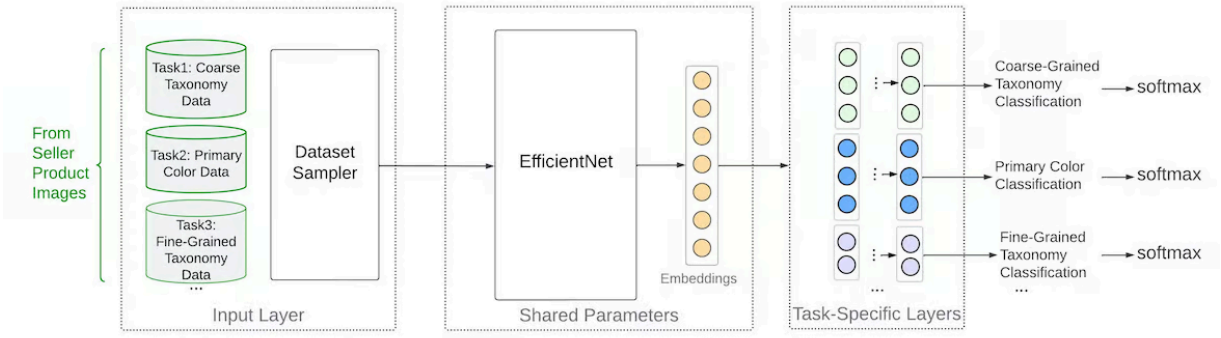
If you'd like to sell your product to US government agencies, then you'll have to follow certain standards and regulations (ITAR, DoD IL4, FedRAMP High and more).



AWS GovCloud is Amazon's cloud computing service for companies that want to follow these standards and build products for the feds.

Slack Engineering published an interesting blog post delving into their experience building on GovCloud and lessons they learned.

slack.engineering/what-we-learned-from-building-govslack



How Etsy built their Search by Image feature

They launched a new feature where you can take a photo of anything and then use that to search for Etsy products. You could take a picture of a certain kind of dinner plate, and Etsy will find similar plates from their 100 million+ items.

This is a fantastic blog post on how they built this feature and the ML techniques they used. A Convolutional Neural Network is used to convert the product images to embeddings vectors. Then, when searching for an item, they convert that item's picture to a vector and look for close vectors in their dataset.

www.etsy.com/codeascraft/from-image-classification-to-multitask-modeling-building-etsys-search-by-image-feature

How Shopify Ensures Consistent Reads

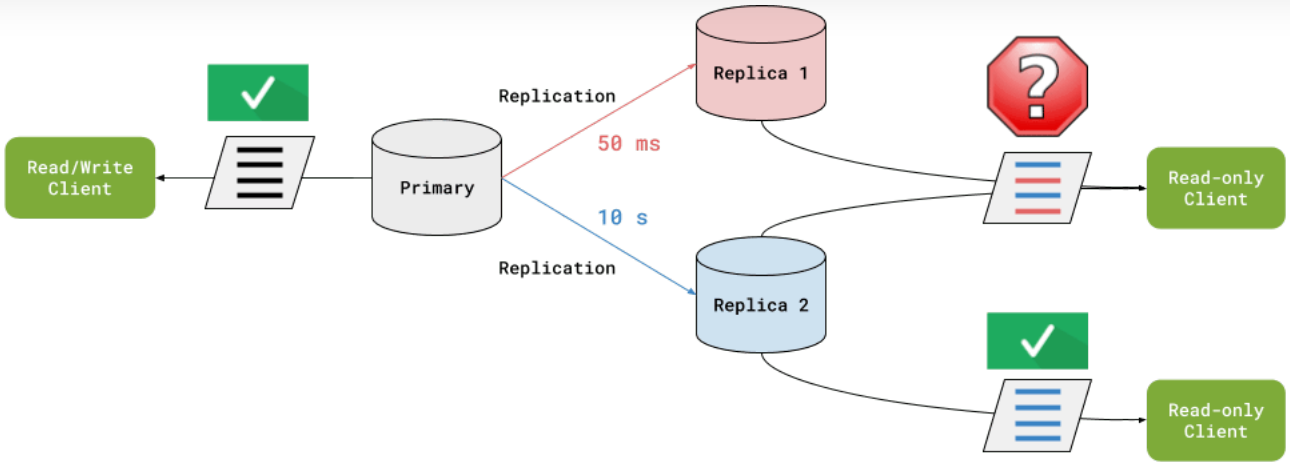
Shopify is an e-commerce platform that helps businesses easily build an online store to sell their products. Over 1.75 million businesses use Shopify and they processed nearly $80 billion dollars in total order value in 2021. In 2022, Shopify merchants will have over 500 million buyers.

For their backend, Shopify relies on a Ruby on Rails monolith with MySQL, Redis and memcached for datastores. If you'd like to read more about their architecture, they gave a good talk about it at InfoQ.

Their MySQL clusters have a read-heavy workload, so they make use of read replicas to scale up. This is where you split your database up into a primary machine and replica machines. The primary database handles write requests (and reads that require strict consistency) while the replicas handle read requests.

An issue with this setup is replication lag. The replica machines will be seconds/a few minutes behind the primary database and will sometimes send back stale data... leading to unpredictability in your application.





Thomas Saunders is a senior software engineer at Shopify, and he wrote a great blog post on how the Shopify team addressed this problem.

Here's a summary

Shopify engineers looked at several possible solutions to solve their consistency issues with their MySQL database replicas.

- Tight Consistency
- Causal Consistency
- Monotonic Read Consistency

Tight Consistency

One method is to enforce tight consistency, where all the replicas are guaranteed to be up to date with the primary server before any other operations are allowed.

In practice, you'll rarely see this implemented because it significantly negates the performance benefits of using replicas. Instead, if you have specific read requests that require strict consistency, then you should just have those executed by the primary machine. Other reads that allow more leeway will go to the replicas.

In terms of stronger consistency guarantees for their other reads (that are handled by replicas), Shopify looked at other approaches.

Causal Consistency

Causal Consistency is where you can specify a read request to go to a replica database that is updated to at least a certain point of time.

So, let's say your application makes a write to the database and later on you have to send a read request that's dependent on that write. With this causal consistency guarantee, you can make a read request that will always go to a database replica that has at least seen that write.

This can be implemented using global transaction identifiers (GTIDs). Every transaction on the primary database will have a GTID associated with it. When the primary database streams changes to the replica machines, the GTIDs associated with those changes will also be sent.

Then, when you send a read request with causal consistency, you'll specify a certain GTID for that read request. Your request will only be routed to read replicas that have at least seen changes up to that GTID.

Shopify considered (and began to implement) this approach in their MySQL clusters, but they found that it would be too complex. Additionally, they didn't really need this for their use cases and they could get by with a weaker consistency guarantee.

Monotonic Read Consistency

With Monotonic read consistency, you have the guarantee that when you make successive read requests, each subsequent read request will go to a database replica that's at least as up-to-date as the replica that served the last read.

This ensures you won't have the moving back in time consistency issue where you could make two database read requests but the second request goes to a replica that has more replication lag than the first. The second query would observe the system state at an earlier point in time than the first query, potentially resulting in a bug.

The easiest way to implement this is to look at any place in your app where you're making multiple sequential reads (that need monotonic read consistency) and route them to the same database replica.

We delve into how Shopify implemented this below.



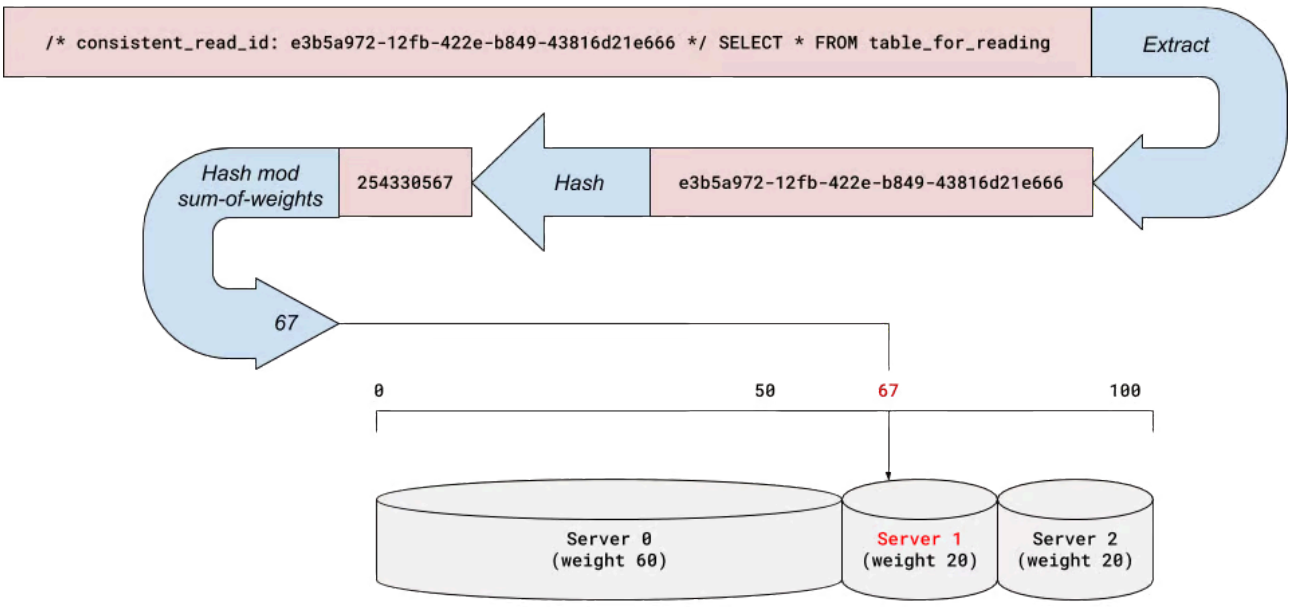
Shopify uses MySQL and application access to the database servers is through a proxy layer provided by ProxySQL.

In order to provide monotonic read consistency, Shopify forked ProxySQL and modified the server selection algorithm.

An application which requires read consistency for a series of requests can give an additional UUID when sending the read requests to the proxy layer.

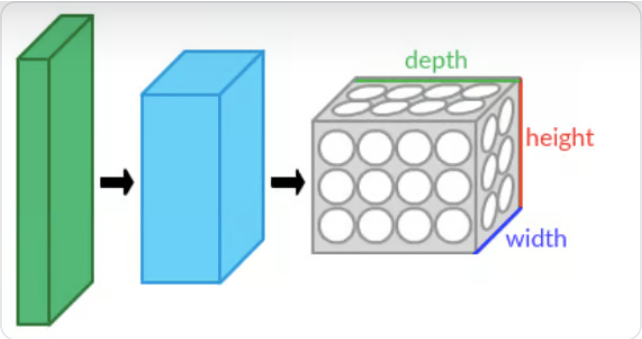
The proxy layer will use that UUID to determine which read replica to send the request to. Read requests with the same UUID will always go to the same read replica.

They hash the UUID and generate an integer and then mod that integer by the sum of all their database replica weights. The resulting answer determines which replica the group of read requests will go to.



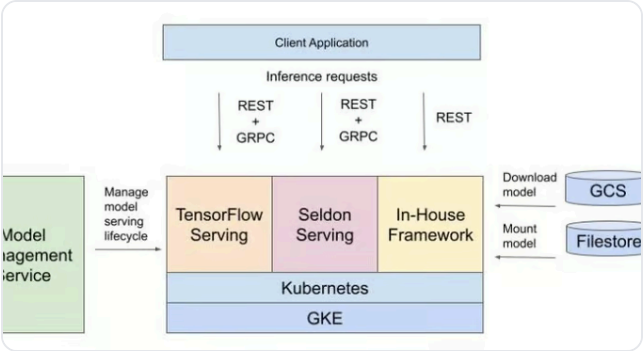
For more details, you can read the full blog post here.

If you'd like to learn more about Consistency Models and the different types of consistency guarantees, I'd highly recommend reading Designing Data Intensive Applications by Martin Kleppman (check out chapter 5 on data replication for this topic).



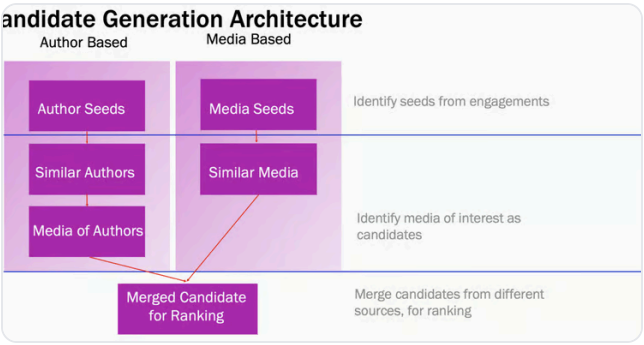
How Dropbox implemented their Image Search feature

Using CNNs, Word Embeddings and an Inverted Index to build Image Search



Redesigning Etsy's Machine Learning Platform

Clean Code's recommendations for Naming. How Tail Call Optimization Works. A primer on Assembly Language. How Etsy redesigned their ML platform and more!



The Engineering Behind Instagram's Recommendations

How Instagram built an Information Retrieval System to power their Suggested Posts feature.

View more ›

Quastor

Get Summaries of Big Tech Engineering Blog Posts on Frontend, Backend, Machine Learning, Data Engineering and more!

HomeAccount Sponsorship

Posts Upgrade Sponsorship

Manage Subscription

Referrals

Enter ` Subscribe

Privacy Policy Terms of Use

Powered by beehiiv