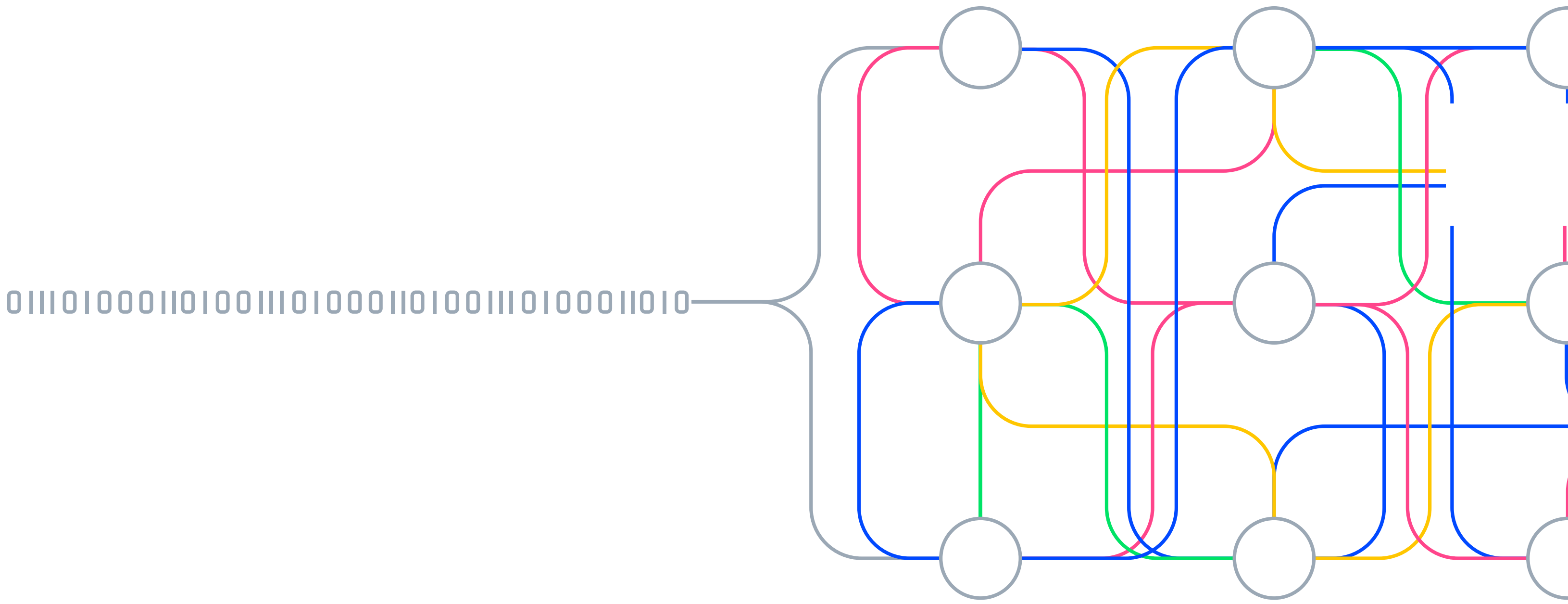
STORJ

How Storing and Distributing AI Training Data and Models Works Using Storj Cloud Object Storage

JT Olio, Jacob Willoughby, Kaloyan Raev, Kevin Leffew



Contents

Introduction 3

Where does storage and distribution fit in the AI workflow?..... 4

LAION training data distribution........ 6

StarCoder model distribution 10

High-performance computing data distribution 13

How does Storj pricing for AI compare? 14

Conclusion 15

Storj and Generative AI: Eight Key Takeaways 16

Introduction

The field of artificial intelligence is going through a transformative explosion. Breakthroughs in the training and use of generative models have led to ongoing seismic shifts in multiple industries. Surprisingly, despite the intense amount of resources needed to train large generative models, the open source community has been holding their own with the behemoth companies that are shifting gears to go all in on AI. In particular, open source generative image models have become the top choice in terms of user scores. Through relatively low resource usage techniques like LoRa, this collaborative environment has produced a massive amount of transfer, storage, and distribution in terms of training data, base models, fine-tuned models and generated output, all over the world, with no real specific base location. This presents a number of storage and distribution challenges and costs for all of this data.

As Prof. Antonin Portelli at the University of Edinburgh discovered for high performance computing data, decentralized storage solutions are appealing precisely in this style of use case. As he says:

> Decentralized storage solutions can in principle allow for a much higher redundancy and geographical spread of the data, allowing strong resilience and wide availability compared to traditional, centralised data centre-based infrastructures

In this paper, we contextualize the different data and distribution-heavy workloads of the generative AI space. Then we share our study of how Storj works for distributing AI training data and generative AI models (along with code for HuggingFace). We will also share similar data and analysis from Dr. Portelli's high performance computing use case. Finally, we share the economics of a Storj implementation.

Where does storage and distribution fit in the AI workflow?

The current generative AI workflow broadly has three main phases:

1. Model training and creation
2. Model customization and fine tuning
3. Model execution and inference

In each of these stages, a compute step processes a large amount of data and runs through the relevant generative AI algorithms. We've laid out these three main phases below:

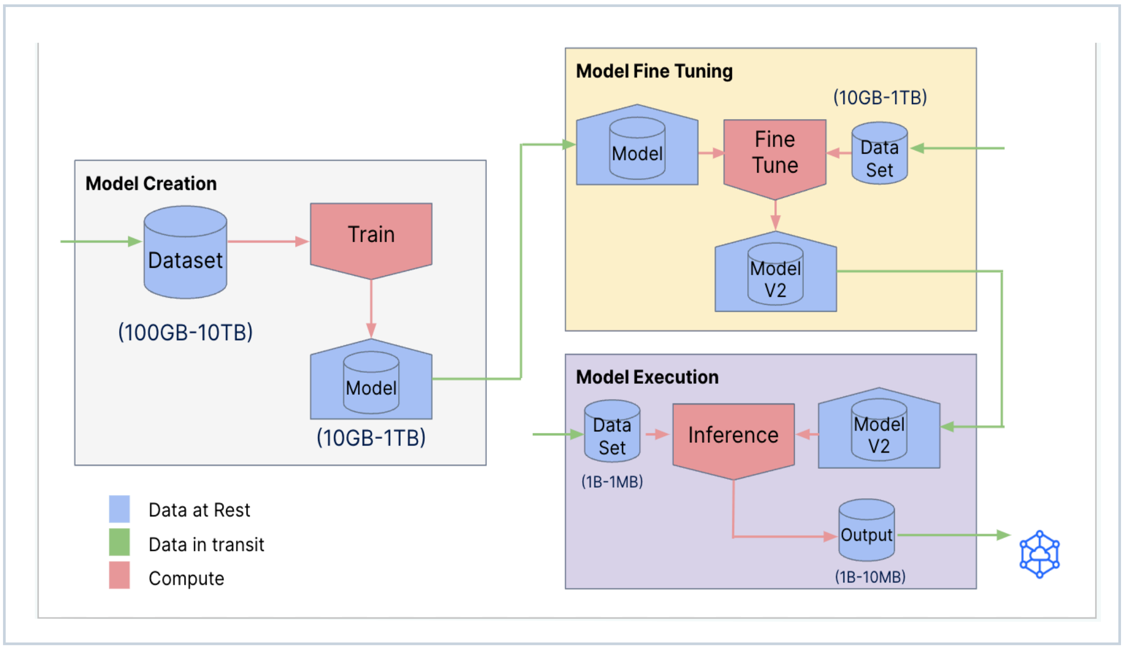


Figure 1: AI Model Development Lifecycle. This figure describes the typical stages of AI model generation, fine tuning and execution. Training data sets typically need to be transferred from a cloud storage repository to a training environment. Data sets and models are transferred between environments and ultimately any output of AI model workloads is delivered over a private network or the public Internet.

In each phase, there is potentially a sizable amount of data both stored and transmitted to compute resources. We have highlighted data as blue, data transfer in green and compute steps as red.

Often, these phases and data sets, especially in the world of open source generative AI, are built and run by different entities and groups, in different locations around the world. This necessarily implies that there is a need to get large data sets distributed securely, performantly, and economically.And this needs to be done with data integrity to many different locations across a broad geographic area, without knowing in advance where those locations may be. In this document we are going to describe dataset distribution (useful for the model creation phase), and model distribution (useful for the execution phase), but there are potentially many other useful implementation patterns.

How Storj works for AI

In the Storj environment, data is not stored in a single location, but is instead distributed across a broad network of independently operated and statistically uncorrelated storage nodes. The system allows the aggregation of underutilized capacity in drives and data centers around the world into a single, logical, S3-compatible object cloud. Every file is encrypted and divided into 64MB segments, which are then sharded using Reed Solomon erasure coding. In a typical deployment, each segment is erasure coded into 80 pieces, of which any 29 can be used to reconstitute the segment. For more detail, please see: https://docs.storj.io/dcs/concepts/overview.

> A storage service that has both fast consistent performance anywhere in the world at a very economical cost.

Since data is stored as small encrypted erasure-encoded pieces of objects distributed over our network of 24,000 points of presence in over 100 countries, the result is a storage service that has both fast consistent performance anywhere in the world at a very economical cost. When it comes to AI workloads, model generation, fine-tuning, and execution take place all over the world, wherever the requisite compute resources are located. Data gravity and data portability have a profound impact on the pace of innovation and the way the Storj network is architected makes it an ideal fit for the distributed nature of AI workload evolution.

LAION training data distribution

Our first case is training dataset distribution.

For test purposes, we downloaded the LAION-5B dataset's English-only embedding, laion-2b-en, which we then uploaded to both Storj and Amazon S3 (us-west-2). While the copy of the data set hosted on Storj[1] represents 3.5 TB of data, we focused our benchmarking and tests on the first ten 1.4 GB image embedding files, 14 GB of data, starting with img_emb_0000.npy[2] and ending with img_emb_0009.npy.

One thing we noted - many of these large data sets are composed of many objects, most of which are likely to be on the upper bound of CDN object delivery maximum sizes[3] . While this presents a challenge for many storage platforms, Storj has no such limitations.

Our test infrastructure was based on seven c-16-intel virtual machines running Ubuntu 22.10 in Digital Ocean, one in each datacenter of sfo3 (San Francisco), ams3 (Amsterdam), fra1 (Frankfurt), syd1 (Sydney), blr1 (Bangalore), tor1 (Toronto), and nyc1 (New York City). We used aria2 1.36.0 for accelerated downloading over HTTP, and we used Storj Uplink release v1.76.2 for Storj's native protocol. To help with statistical accuracy and testing, we used the Hyperfine benchmarking tool release v1.16.1.

Note that the map provided on each file on the Storj platform shares the real location of the storage nodes hosting the data, geolocated to their closest city. Below are the real locations for this first test file:

1 LAION-5B English embedding dataset hosted on Storj: https://link.storjshare.io/s/jvdbhbogkj5x-vd4occnk5c5j3nya/datasets/laion5b/embeddings/laion2B-en/

2 Full URL: https://link.storjshare.io/s/jvdbhbogkj5xvd4occnk5c5j3nya/datasets/laion5b/embeddings/laion2B-en/img_emb/img_emb_0000.npy

3 Amazon CloudFront's largest object size is 30GB. Other providers have similar restrictions.

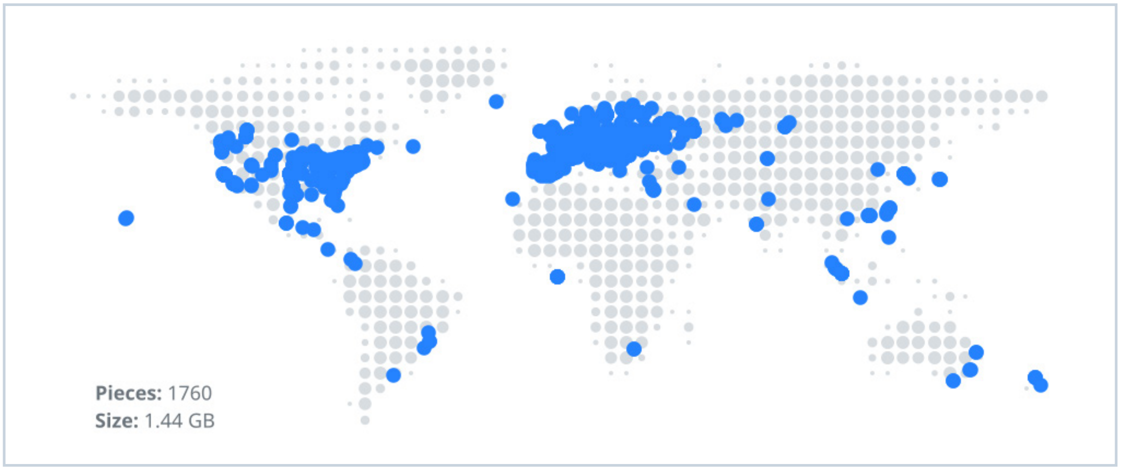


Figure 2: Storj Object Map. This image shows the approximate location of Storj points of presence storing 1,760 encrypted erasure-encoded pieces of a 1.44GB object. Only a subset of the nearest pieces are required to reconstitute the original object.

For each of the first 10 image embedding files, on each virtual machine, hyperfine was run as follows:

```
hyperfine –cleanup 'rm /tmp/dobench-* || true' –show-output \

  –export-json benchmark-REGION-TIMESTAMP.json \

  –command-name "AWS S3 us-west-2" "aria2c -x 16 -s 16 https://bucket.s3.us-west-2.amazonaws.com/…/img_emb/img_emb_0000.npy dir=/ –out=tmp/dobench-out" \

  –command-name "AWS S3-acceleration us-west-2" "aria2c -x 16 -s 16 https://bucket.s3-accelerate.amazonaws.com/…/img_emb/img_emb_0000.npy –dir=/ –out=tmp/dobench-out" \

  –command-name "Storj DCS Global" "uplink cp -p 16 sj://bucket/img_emb/img_emb_0000.npy /tmp/dobench-out" \

  –command-name "Storj HTTP Gateway" "aria2c -x 16 -s 16 https://link.storjshare.io/raw/…/img_emb/img_emb_0000.npy –dir=/ –out=tmp/dobench-out"
```

This command tests and benchmarks standard S3 access, S3 access with transfer acceleration, Storj direct network access, and Storj access through our HTTP gateway.

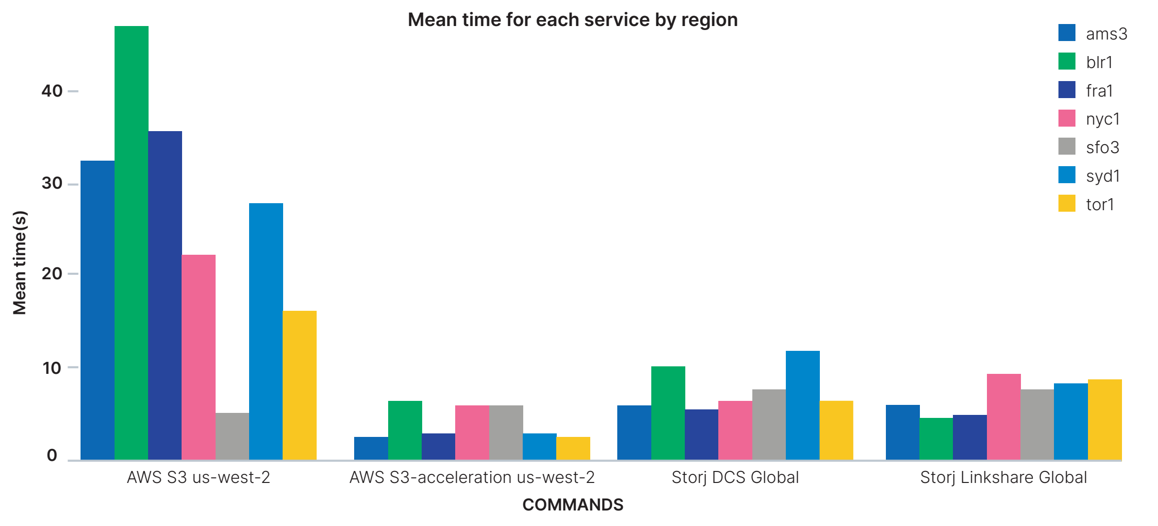
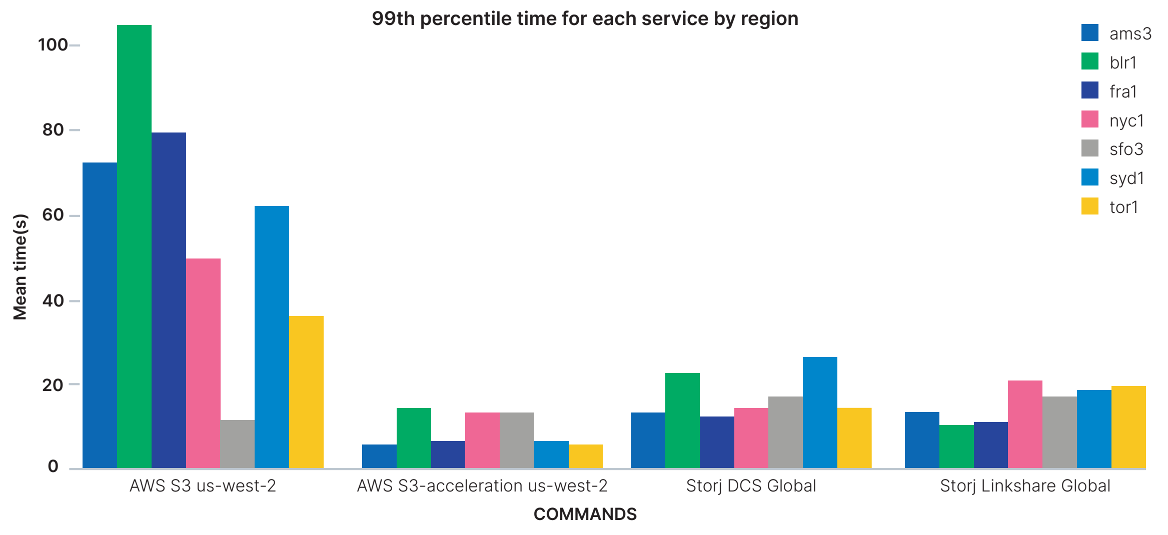


Figure 3: Mean Performance Comparison AWS-Storj. This diagram shows a comparison of the mean time to download the single upload of the test data test data set of ten 1.44GB image objects from the seven different datacenters in different regions, comparing the performance of standard S3, AWS S3 Accelerated Transfer, Storj native protocol and Storj Edge services. Lower values indicate faster overall performance.

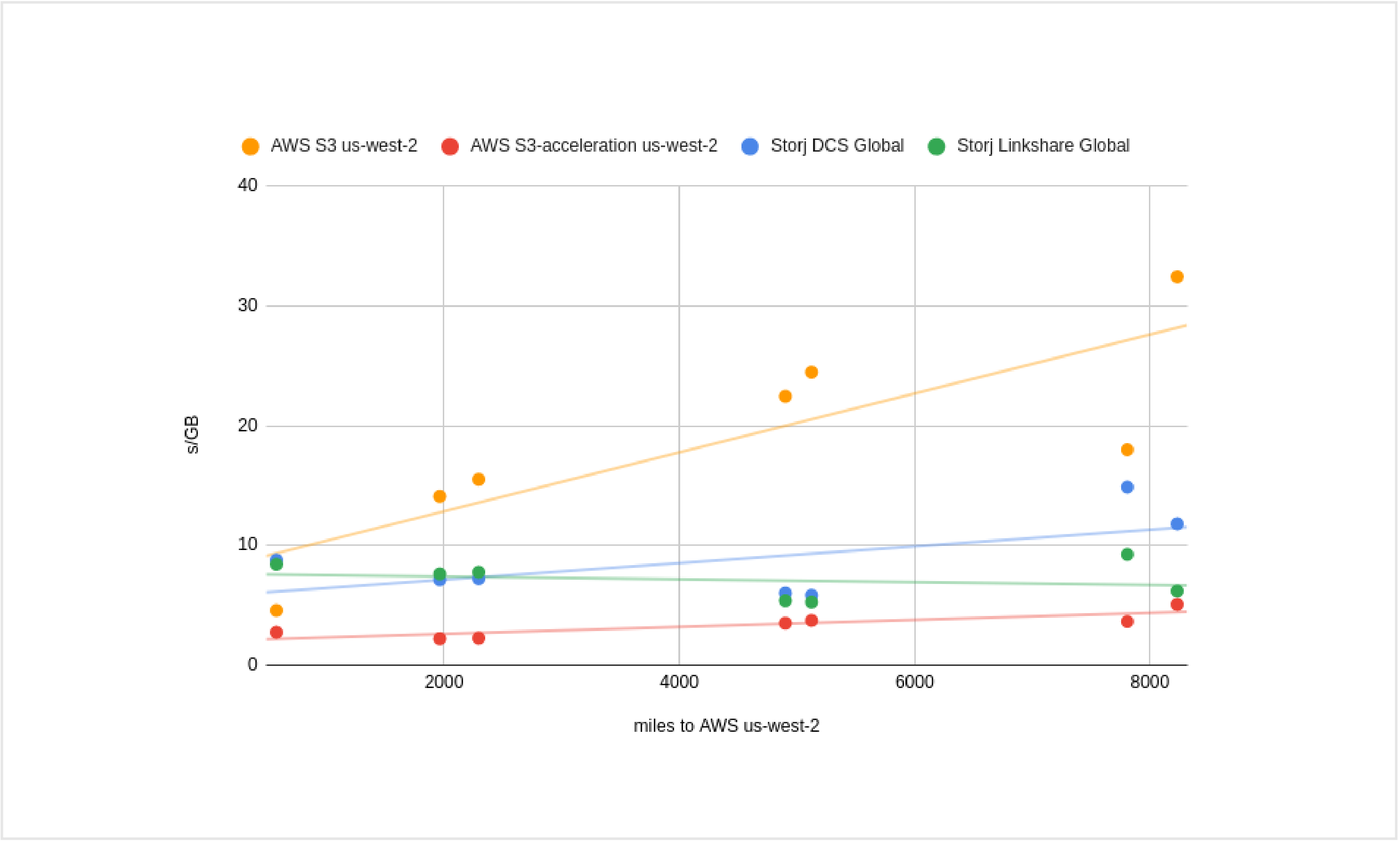
Figure 4: 99th Percentile Performance Comparison AWS-Storj. This diagram shows a comparison of the 99th percentile time to download the single upload of the test data test data



set of ten 1.44GB image objects from the seven different datacenters in different regions, comparing the performance of standard S3, AWS S3 Accelerated Transfer, Storj native protocol and Storj Edge services. High variability indicates inconsistent global performance.

The results are that standard Storj performance in general is much better than standard S3 performance, considering global distribution. Here is the same data plotted as distance from the AWS us-west-2 datacenter:

Figure 5: Global Performance Consistency Comparison AWS-Storj. The Figure above shows the change in performance between download samples based on progressively increasing geographic distance from the point of origin, using geographic distance as a proxy for number of



hops for network transit. An increasing slope indicates a degradation in performance based on globally distributed access patterns, whereas a flat line indicates globally consistent performance regardless of origin or access pattern.

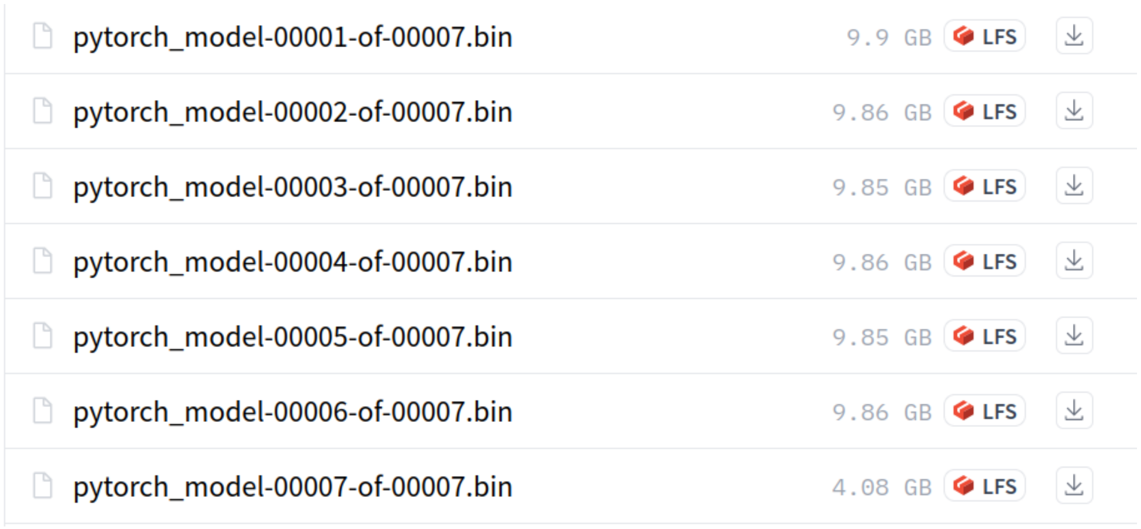
Try it out for yourself and download LAION-5B's English language embedding here: https://link.storjshare.io/s/jvdbhbogkj5xvd4occnk5c5j3nya/datasets/laion5b/embeddings/laion2B-en/

StarCoder model distribution

Our next assessment is model distribution.

We conducted a real-life experiment by downloading the relatively large (64 GB) StarCoder AI model from the HuggingFace. We executed the experiment on a Hetzner cpx31 VM at Ashburn, Virginia.

As we can see on the HuggingFace page of the StarCoder model, all large files are stored in Git-LFS.



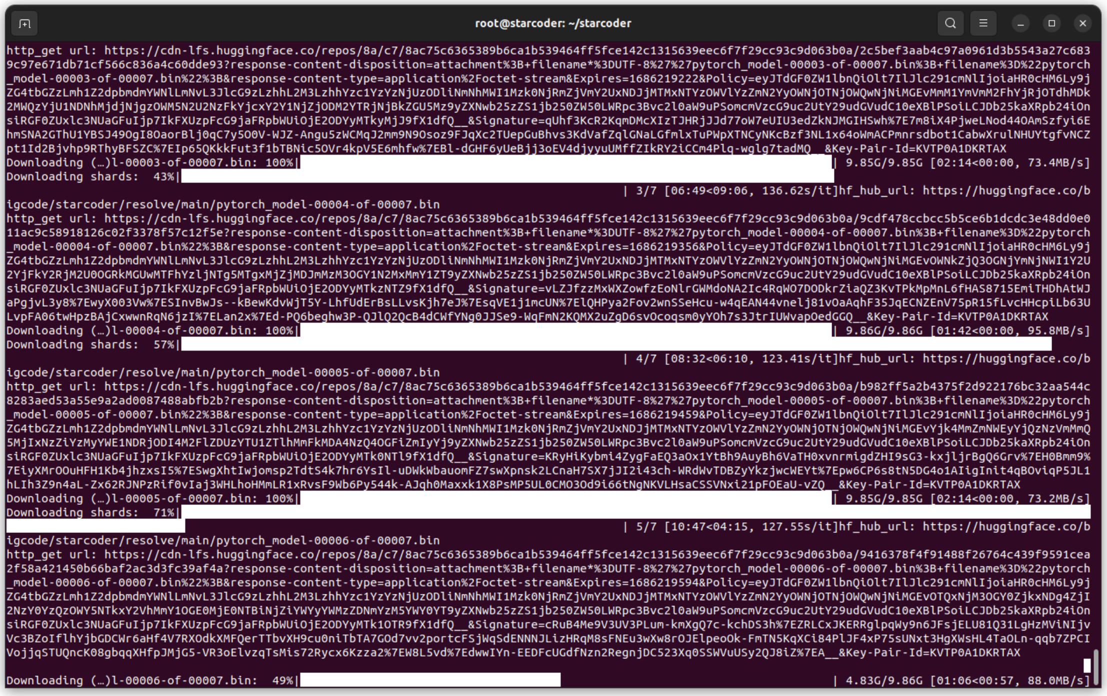
pytorch_model-00001-of-00007.bin	9.9 GB	LFS
pytorch_model-00002-of-00007.bin	9.86 GB	LFS
pytorch_model-00003-of-00007.bin	9.85 GB	LFS
pytorch_model-00004-of-00007.bin	9.86 GB	LFS
pytorch_model-00005-of-00007.bin	9.85 GB	LFS
pytorch_model-00006-of-00007.bin	9.86 GB	LFS
pytorch_model-00007-of-00007.bin	4.08 GB	LFS

Figure 6: Sample Training Set Object Sizes. The image above shows a representative sample of the objects that compose the StarCoder AI model.

First, we modified the `http_get` and `hf_hub_url` functions of the `file_download.py` file of the `huggingface_hub library` to print the URL of the downloaded file. Then we executed a Python script for code generation that downloaded the entire model through the `huggingface_hub` library.

The screenshot below shows that downloads for files stored on Git-LFS are redirected to `cdn-lfs.huggingface.co`. The average download speed for all 64 GB of data was 75.8 MB/s.

Figure 7. Actual Download Performance - AWS S3. Screen capture showing sample of actual



```
root@starcoder: ~/starcoder
http_get url: https://cdn-lfs.huggingface.co/repos/8a/c7/8ac75c6365389b6ca1b539464ff5fce142c1315639eec6f7f29cc93c9d063b0a/2c5bef3aab4c97a0961d3b5543a27c683
9c97e671db71cf566c836a4c60dde93?response-content-disposition=attachment%3B+filename*%3DUTF-8%27%27pytorch_model-00003-of-00007.bin%3B+filename%3D%22pytorch
_model-00003-of-00007.bin%22%3B&response-content-type=application%2Foctet-stream&Expires=1686219222&Policy=eyJTdGF0ZW1lbnQiOlt7IlJlc291cmNlIjoiaHR0cHM6Ly9j
ZG4tbGZzLmh1Z2dpbmdmYWNlLmNvL3JlcG9zLzhhL2M3LzhhYzc1YzYzNjUzODliNmNhMWI1Mzk0NjRmZjVmY2UxNDJjMTMxNTYzOWVlYzZmN2YyOWNjOTNjOWQwNjNiMGEvMmM1YmVmM2FhYjRjOTdhMDk
2MWQzYjU1NDNhMjdjNjgzOWM5N2U2NzFkYjcxY2Y1NjZjODM2YTRjNjBkZGU5Mz9yZXNwb25zZS1jb250ZW50LWRpc3Bvc2l0aW9uPSomcmVzcG9uc2UtY29udGVudC10eXBlPSoiLCJDb25kaXRpb24iOn
siRGF0ZUxlc3NUaGFuIjp7IkFXUzpFcG9jaFRpbWUiOjE2ODYyMTkyMjJ9fX1dfQ__&Signature=qUhf3KcR2KqmDMcXIzTJHRjJJd77oW7eUIU3edZkNJMGIHSwh%7E7m8iX4PjweLNod44OAmSzfyi6E
hmSNA2GThU1YBSJ49OgI8OaorBlj0qC7y5O0V-WJZ-Angu5zWCMqJ2mm9N9Osoz9FJqXc2TUepGuBhvs3KdVafZqlGNaLGfmlxTuPWpXTNCyNKcBzf3NL1x64oWmACPmnrsdbot1CabwXrulNHUYtgfvNCZ
pt1Id2Bjvhp9RThyBFSZC%7EIp65QKkkFut3f1bTBNic5OVr4kpV5E6mhfw%7EBl-dGHF6yUeBjj3oEV4djyyuUMffZIkRY2iCCm4Plq-wglg7tadMQ__&Key-Pair-Id=KVTP0A1DKRTAX
Downloading (…)l-00003-of-00007.bin: 100%|                                                                      | 9.85G/9.85G [02:14<00:00, 73.4MB/s]
Downloading shards:  43%|
                                                                                          | 3/7 [06:49<09:06, 136.62s/it]hf_hub_url: https://huggingface.co/b
igcode/starcoder/resolve/main/pytorch_model-00004-of-00007.bin
http_get url: https://cdn-lfs.huggingface.co/repos/8a/c7/8ac75c6365389b6ca1b539464ff5fce142c1315639eec6f7f29cc93c9d063b0a/9cdf478ccbcc5b5ce6b1dcdc3e48dd0e0
11ac9c58918126c02f3378f57c12f5e?response-content-disposition=attachment%3B+filename*%3DUTF-8%27%27pytorch_model-00004-of-00007.bin%3B+filename%3D%22pytorch
_model-00004-of-00007.bin%22%3B&response-content-type=application%2Foctet-stream&Expires=1686219356&Policy=eyJTdGF0ZW1lbnQiOlt7IlJlc291cmNlIjoiaHR0cHM6Ly9j
ZG4tbGZzLmh1Z2dpbmdmYWNlLmNvL3JlcG9zLzhhL2M3LzhhYzc1YzYzNjUzODliNmNhMWI1Mzk0NjRmZjVmY2UxNDJjMTMxNTYzOWVlYzZmN2YyOWNjOTNjOWQwNjNiMGEvOWNkZjQ3OGNjYmNjNWI1Y2U
2YjFkY2RjM2U0OGRkMGUwMTFhYzljNTg5MTgxMjZjMDJmMzM3OGY1N2MxMmY1ZT9yZXNwb25zZS1jb250ZW50LWRpc3Bvc2l0aW9uPSomcmVzcG9uc2UtY29udGVudC10eXBlPSoiLCJDb25kaXRpb24iOn
siRGF0ZUxlc3NUaGFuIjp7IkFXUzpFcG9jaFRpbWUiOjE2ODYyMTkzNTZ9fX1dfQ__&Signature=vLZJfzzMxWXZowfzEoNlrGWMdoNA2Ic4RqWO7DODkrZiaQZ3KvTPkMpMnL6fHAS8715EmiTHDhAtWJ
aPgjvL3y8%7EwyX003Vw%7ESInvBwJs--kBewKdvWjT5Y-LhfUdErBsLLvsKjh7eJ%7EsqVE1j1mcUN%7ElQHPya2Fov2wnSSeHcu-w4qEAN44vnelj81vOaAqhF35JqECNZEnV75pR15fLvcHHcpiLb63U
LvpFA06twHpzBAjCxwwnRqN6jzI%7ELan2x%7Ed-PQ6beghw3P-QJlQ2QcB4dCWfYNg0JJSe9-WqFmN2KQMX2uZgD6svOcoqsm0yYOh7s3JtrIUWvapOedGGQ__&Key-Pair-Id=KVTP0A1DKRTAX
Downloading (…)l-00004-of-00007.bin: 100%|                                                                      | 9.86G/9.86G [01:42<00:00, 95.8MB/s]
Downloading shards:  57%|
                                                                                          | 4/7 [08:32<06:10, 123.41s/it]hf_hub_url: https://huggingface.co/b
igcode/starcoder/resolve/main/pytorch_model-00005-of-00007.bin
http_get url: https://cdn-lfs.huggingface.co/repos/8a/c7/8ac75c6365389b6ca1b539464ff5fce142c1315639eec6f7f29cc93c9d063b0a/b982ff5a2b4375f2d922176bc32aa544c
8283aed53a55e9a2ad0087488abfb2b?response-content-disposition=attachment%3B+filename*%3DUTF-8%27%27pytorch_model-00005-of-00007.bin%3B+filename%3D%22pytorch
_model-00005-of-00007.bin%22%3B&response-content-type=application%2Foctet-stream&Expires=1686219459&Policy=eyJTdGF0ZW1lbnQiOlt7IlJlc291cmNlIjoiaHR0cHM6Ly9j
ZG4tbGZzLmh1Z2dpbmdmYWNlLmNvL3JlcG9zLzhhL2M3LzhhYzc1YzYzNjUzODliNmNhMWI1Mzk0NjRmZjVmY2UxNDJjMTMxNTYzOWVlYzZmN2YyOWNjOTNjOWQwNjNiMGEvYjk4MmZmNWEyYjQzNzVmMmQ
5MjIxNzZiYzMyYWE1NDRjODI4M2FlZDUzYTU1ZTlhMmFkMDA4NzQ4OGFiZmIyYj9yZXNwb25zZS1jb250ZW50LWRpc3Bvc2l0aW9uPSomcmVzcG9uc2UtY29udGVudC10eXBlPSoiLCJDb25kaXRpb24iOn
siRGF0ZUxlc3NUaGFuIjp7IkFXUzpFcG9jaFRpbWUiOjE2ODYyMTk0NTl9fX1dfQ__&Signature=KRyHiKybmi4ZygFaEQ3aOx1YtBh9AuyBh6VaTH0xvnrmigdZHI9sG3-kxjljrBgQ6Grv%7EH0Bmm9%
7EiyXMrOOuHFH1Kb4jhzxsI5%7ESwgXhtIwjomsp2TdtS4k7hr6YsIl-uDWkWbauomFZ7swXpnsk2LCnaH7SX7jJI2i43ch-WRdWvTDBZyYkzjwcWEYt%7Epw6CP6s8tN5DG4o1AIigInit4qBOviqP5JL1
hLIh3Z9n4aL-Zx62RJNPzRif0vIaj3WHLhoHMmLR1xRvsF9Wb6Py544k-AJqh0Maxxk1X8PsMP5UL0CMO3Od9i66tNgNKVLHsaCSSVNxi21pFOEaU-vZQ__&Key-Pair-Id=KVTP0A1DKRTAX
Downloading (…)l-00005-of-00007.bin: 100%|                                                                      | 9.85G/9.85G [02:14<00:00, 73.2MB/s]
Downloading shards:  71%|
                                                                                          | 5/7 [10:47<04:15, 127.55s/it]hf_hub_url: https://huggingface.co/b
igcode/starcoder/resolve/main/pytorch_model-00006-of-00007.bin
http_get url: https://cdn-lfs.huggingface.co/repos/8a/c7/8ac75c6365389b6ca1b539464ff5fce142c1315639eec6f7f29cc93c9d063b0a/9416378f4f91488f26764c439f9591cea
2f58a421450b66baf2ac3d3fc39af4a?response-content-disposition=attachment%3B+filename*%3DUTF-8%27%27pytorch_model-00006-of-00007.bin%3B+filename%3D%22pytorch
_model-00006-of-00007.bin%22%3B&response-content-type=application%2Foctet-stream&Expires=1686219594&Policy=eyJTdGF0ZW1lbnQiOlt7IlJlc291cmNlIjoiaHR0cHM6Ly9j
ZG4tbGZzLmh1Z2dpbmdmYWNlLmNvL3JlcG9zLzhhL2M3LzhhYzc1YzYzNjUzODliNmNhMWI1Mzk0NjRmZjVmY2UxNDJjMTMxNTYzOWVlYzZmN2YyOWNjOTNjOWQwNjNiMGEvOTQxNjM3OGY0ZjkxNDg4ZjI
2NzY0YzQzOWY5NTkxY2VhMmY1OGE0MjE0NTBiNjZiYWYyYWMzZDNmYzM5YWY0YT9yZXNwb25zZS1jb250ZW50LWRpc3Bvc2l0aW9uPSomcmVzcG9uc2UtY29udGVudC10eXBlPSoiLCJDb25kaXRpb24iOn
siRGF0ZUxlc3NUaGFuIjp7IkFXUzpFcG9jaFRpbWUiOjE2ODYyMTk1OTR9fX1dfQ__&Signature=cRuB4Me9V3UV3PLum-kmXgQ7c-kchDS3h%7EZRLCxJKERRglpqWy9n6JFsjELU81Q31LgHzMViNIjv
Vc3BZoIflhYjbGDCWr6aHf4V7RXOdkXMFQerTTbvXH9cu0niTbTA7GOd7vv2portcFSjWqSdENNNJLizHRqM8sFNEu3wXw8rOJElpeoOk-FmTN5KqXCi84PlJF4xP75sUNxt3HgXWsHL4TaOLn-qqb7ZPCI
VojjqSTUQncK08gbqqXHfpJMjG5-VR3oElvzqTsMis72Rycx6Kzza2%7EW8L5vd%7EdwwIYn-EEDFcUGdfNzn2RegnjDC523Xq0SSWVuUSy2QJ8iZ%7EA__&Key-Pair-Id=KVTP0A1DKRTAX

Downloading (…)l-00006-of-00007.bin:  49%|                                                                      | 4.83G/9.86G [01:06<00:57, 88.0MB/s]
```

throughput performance of downloads from AWS S3 with access credentials obscured.

It's worth noting that the download happens from an AWS S3 bucket through CloudFront, but the CloudFront cache is not hydrated yet.

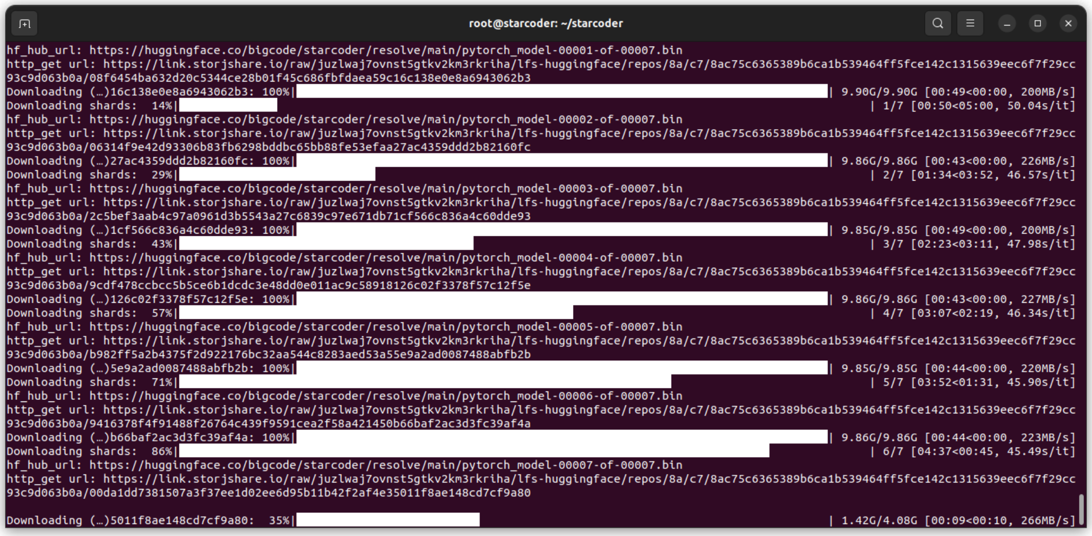
To conduct a download of the same model from Storj, we executed the following steps:

- We replicated the Git-LFS structure of the StarCoder model in a Storj bucket: https://link.storjshare.io/s/juzlwaj7ovnst5gtkv2km3rkriha/lfs-huggingface
- We deleted the local download cache of the huggingface_hub library.
- We modified the `get_hf_file_metadata` function to override the redirect for Git-LFS downloads to `link.storjshare.io` instead of `cdn-lfs.huggingface.co`.
- We modified the http_get function to download from link.storjshare.io with 16 parallel connections to get the best from the decentralized nature of the Storj network.

With these changes, we downloaded the 64 GB model with an average speed of 212.7 MB/s, almost 3x an improvement over the original rate (75.8 MB/s).

Figure 8. Actual Download Performance - Storj. Screen capture showing sample of actual throughput performance of downloads from Storj with access credentials obscured.

We published our modifications to the `huggingface_hub` library as a Python



```
root@starcoder: ~/starcoder
hf_hub_url: https://huggingface.co/bigcode/starcoder/resolve/main/pytorch_model-00001-of-00007.bin
http_get url: https://link.storjshare.io/raw/juzlwaj7ovnst5gtkv2km3rkriha/lfs-huggingface/repos/8a/c7/8ac75c6365389b6ca1b539464ff5fce142c1315639eec6f7f29cc
93c9d063b0a/08f6454ba632d20c5344ce28b01f45c686fbfdaea59c16c138e0e8a6943062b3
Downloading (…)16c138e0e8a6943062b3: 100%|                    | 9.90G/9.90G [00:49<00:00, 200MB/s]
Downloading shards:  14%|                                       | 1/7 [00:50<05:00, 50.04s/it]
hf_hub_url: https://huggingface.co/bigcode/starcoder/resolve/main/pytorch_model-00002-of-00007.bin
http_get url: https://link.storjshare.io/raw/juzlwaj7ovnst5gtkv2km3rkriha/lfs-huggingface/repos/8a/c7/8ac75c6365389b6ca1b539464ff5fce142c1315639eec6f7f29cc
93c9d063b0a/06314f9e42d93306b83fb6298bddbc65bb88fe53efaa27ac4359ddd2b82160fc
Downloading (…)27ac4359ddd2b82160fc: 100%|                    | 9.86G/9.86G [00:43<00:00, 226MB/s]
Downloading shards:  29%|                                       | 2/7 [01:34<03:52, 46.57s/it]
hf_hub_url: https://huggingface.co/bigcode/starcoder/resolve/main/pytorch_model-00003-of-00007.bin
http_get url: https://link.storjshare.io/raw/juzlwaj7ovnst5gtkv2km3rkriha/lfs-huggingface/repos/8a/c7/8ac75c6365389b6ca1b539464ff5fce142c1315639eec6f7f29cc
93c9d063b0a/2c5bef3aab4c97a0961d3b5543a27c6839c97e671db71cf566c836a4c60dde93
Downloading (…)1cf566c836a4c60dde93: 100%|                    | 9.85G/9.85G [00:49<00:00, 200MB/s]
Downloading shards:  43%|                                       | 3/7 [02:23<03:11, 47.98s/it]
hf_hub_url: https://huggingface.co/bigcode/starcoder/resolve/main/pytorch_model-00004-of-00007.bin
http_get url: https://link.storjshare.io/raw/juzlwaj7ovnst5gtkv2km3rkriha/lfs-huggingface/repos/8a/c7/8ac75c6365389b6ca1b539464ff5fce142c1315639eec6f7f29cc
93c9d063b0a/9cdf478ccbcc5b5ce6b1dcdc3e48dd0e011ac9c58918126c02f3378f57c12f5e
Downloading (…)126c02f3378f57c12f5e: 100%|                    | 9.86G/9.86G [00:43<00:00, 227MB/s]
Downloading shards:  57%|                                       | 4/7 [03:07<02:19, 46.34s/it]
hf_hub_url: https://huggingface.co/bigcode/starcoder/resolve/main/pytorch_model-00005-of-00007.bin
http_get url: https://link.storjshare.io/raw/juzlwaj7ovnst5gtkv2km3rkriha/lfs-huggingface/repos/8a/c7/8ac75c6365389b6ca1b539464ff5fce142c1315639eec6f7f29cc
93c9d063b0a/b982ff5a2b4375f2d922176bc32aa544c8283aed53a55e9a2ad0087488abfb2b
Downloading (…)5e9a2ad0087488abfb2b: 100%|                    | 9.85G/9.85G [00:44<00:00, 220MB/s]
Downloading shards:  71%|                                       | 5/7 [03:52<01:31, 45.90s/it]
hf_hub_url: https://huggingface.co/bigcode/starcoder/resolve/main/pytorch_model-00006-of-00007.bin
http_get url: https://link.storjshare.io/raw/juzlwaj7ovnst5gtkv2km3rkriha/lfs-huggingface/repos/8a/c7/8ac75c6365389b6ca1b539464ff5fce142c1315639eec6f7f29cc
93c9d063b0a/9416378f4f91488f26764c439f9591cea2f58a421450b66baf2ac3d3fc39af4a
Downloading (…)b66baf2ac3d3fc39af4a: 100%|                    | 9.86G/9.86G [00:44<00:00, 223MB/s]
Downloading shards:  86%|                                       | 6/7 [04:37<00:45, 45.49s/it]
hf_hub_url: https://huggingface.co/bigcode/starcoder/resolve/main/pytorch_model-00007-of-00007.bin
http_get url: https://link.storjshare.io/raw/juzlwaj7ovnst5gtkv2km3rkriha/lfs-huggingface/repos/8a/c7/8ac75c6365389b6ca1b539464ff5fce142c1315639eec6f7f29cc
93c9d063b0a/00da1dd7381507a3f37ee1d02ee6d95b11b42f2af4e35011f8ae148cd7cf9a80

Downloading (…)5011f8ae148cd7cf9a80:  35%|                    | 1.42G/4.08G [00:09<00:10, 266MB/s]
```

monkey patch, so others can quickly reproduce these results: https://github.com/storj/huggingface-hub-storj-patch

We also have a recorded demo of this experiment: https://link.storjshare.io/s/jxlzyjgj3pdc2msqm5my4w4pv3qa/videos/storj-huggingface-poc-demo.mp4

High-performance computing data distribution

Storj's use cases are not limited to AI training data sets and model distribution.

In 2021, Prof. Antonin Portelli at the University of Edinburgh conducted an experiment to ascertain the suitability of decentralized storage via Storj for high performance computing data storage and scientific data sharing.

The results were fascinating, and are worth reading in detail in his original report, along with the updated statistics from May of 2023.

One thing he focused on is how the Storj network adapts to concurrency and parallelism. Below is a graph that shows the throughput with various concurrency configurations:

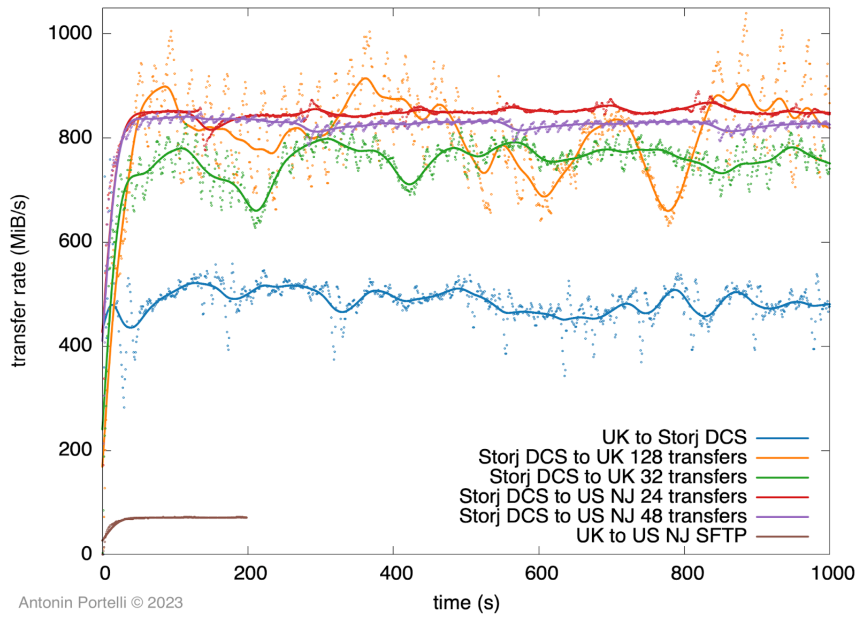


Figure 9. Sample Result Download Performance - Storj. The Figure above shows the actual test results of object downloads between Storj and data centers located in Edinburgh, Scotland and New Jersey from the May, 2023 updated performance test conducted by Prof. Antonin Portelli.

To restate his key takeaway from his 2023 update: You can really reach very fast transfer rates thanks to parallelism. And this is something which is built-in natively in the Storj network, which is really nice because the data from many nodes are scattered all over the world. So you can really expect a good buildup of performances.

How does Storj pricing for AI compare?

As you've seen, Storj offers durable storage with S3 Transfer Accelerated-like speeds. But even more excitingly, Storj is able to do it with the lowest prices in the cloud market.

To drive that last point home, here are two price comparisons:

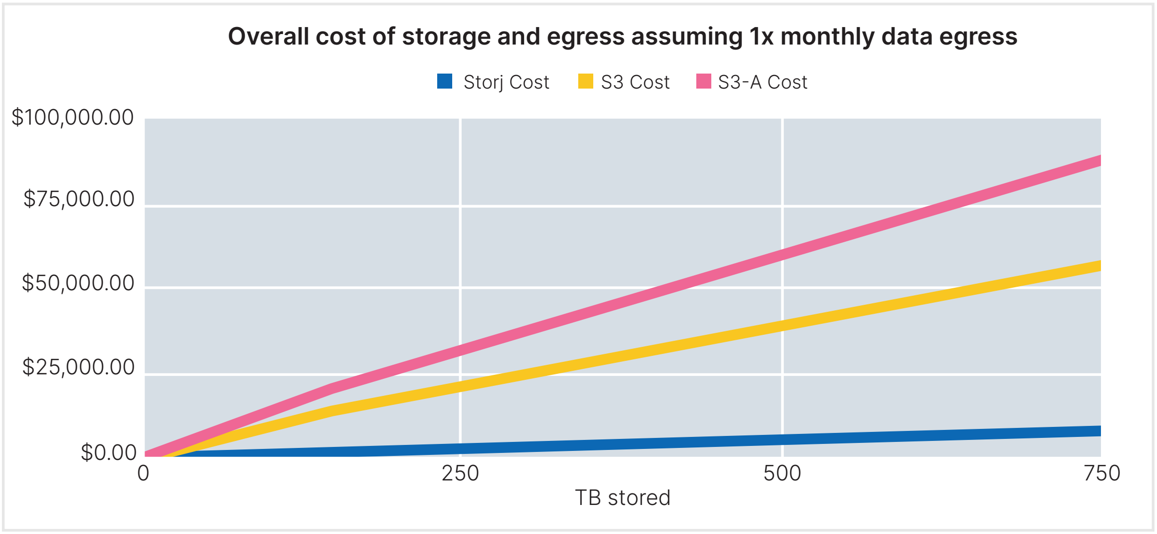


Figure 10: Cost Comparison AWS-Storj. The figure above shows the one month cost to store and retrieve a data set 1x based on the standard retail price published on the providers' publicly available pricing web page.

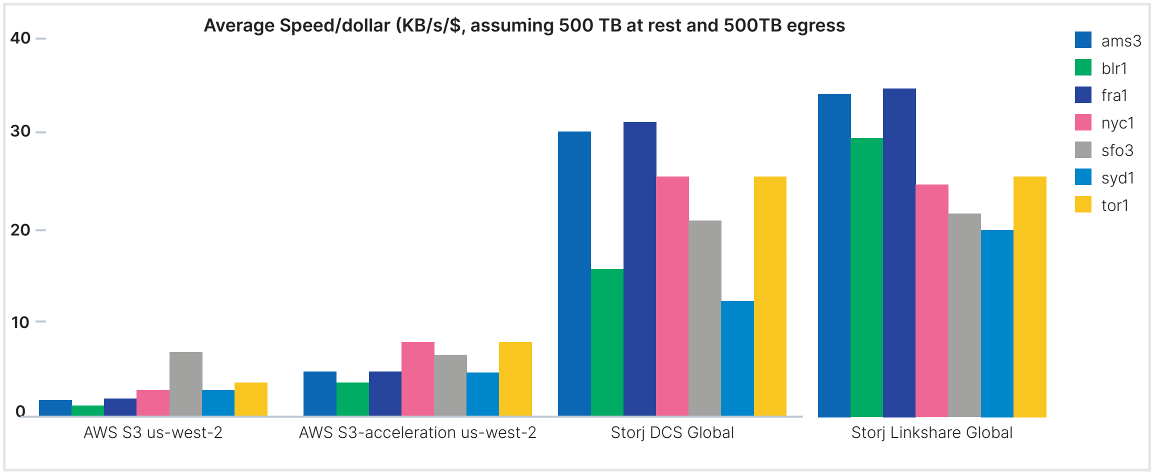


Figure 11: KB/s/$ Cost Comparison AWS-Storj. The figure above shows a normalized KB/s/$ cost to store and retrieve a 500TB data set 1x based on the standard retail price published on the providers' publicly available pricing web page. Lower values indicate a high cost as performance and corresponding throughput increases.

Conclusion

The number of AI data sets is growing exponentially and the size of the data sets is increasing rapidly as training models become increasingly sophisticated. Open source models and innovative startups have been able to outpace development by the largest and best funded hyperscale platforms on the market. As the market evolves the sheer expense of storing and moving large data sets around the internet may allow those same hyperscale platforms to create a moat around the generative AI use case, stifling innovation by artificially driving up the cost to store and transfer data as a barrier to entry.

For global file distribution of large amounts of data, such as AI training data, AI models, any other point of the generative AI workflow, or even other use cases entirely, Storj is able to distribute globally at AWS S3 Transfer Accelerated-like speeds with the lowest prices in the cloud market.

Storj is also easy to integrate into existing applications, either with native language bindings, or with hosted gateway solutions. Storj does not require changes in application behavior and can be used in any context existing cloud object storage solutions are used.

We hope you found this analysis interesting! We would like to help you improve your model distribution strategy utilizing parallelism + erasure coding. Looking forward to working with you to replicate these findings.

Storj and Generative AI: Eight Key Takeaways

1. Storing and distributing large amounts of AI training data, models, and output can present storage and distribution challenges. Distributed storage solutions, such as Storj, can offer higher redundancy, geographical spread, and resilience compared to traditional data center-based infrastructures.

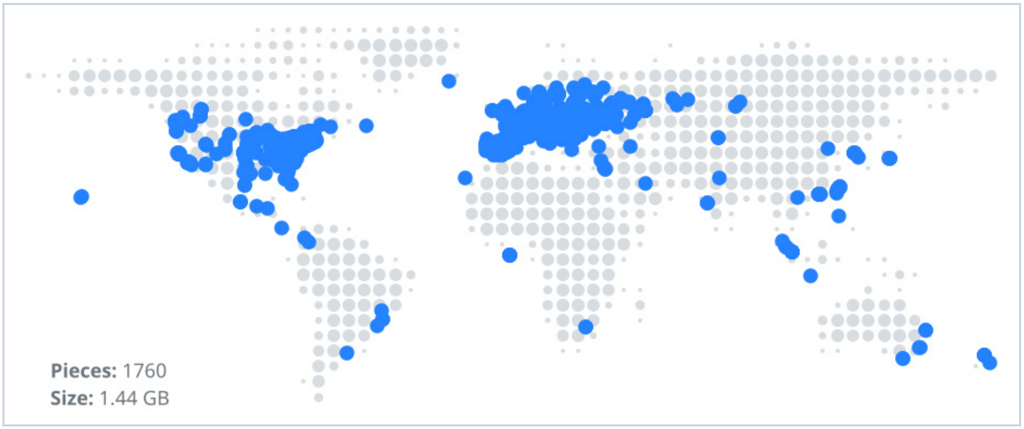


Figure 1: Storj Object Map. This image shows the approximate location of Storj points of presence storing 1,760 encrypted erasure-encoded pieces of a 1.44GB object. Only a subset of the nearest pieces are required to reconstitute the original object.

2. The generative AI workflow broadly has three main phases: model training and creation, model customization and finetuning, and model execution and inference. Each of these phases requires a large amount of data processing.

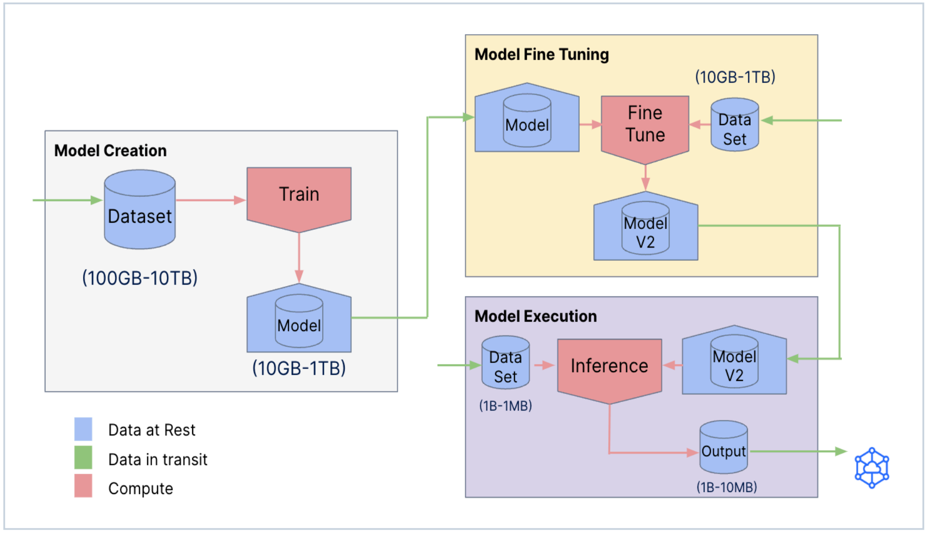


Figure 2: AI Model Development Lifecycle. This figure describes the typical stages of AI model generation, fine tuning and execution. Training data sets typically need to be transferred from a cloud storage repository to a training environment. Data sets and models are transferred between environments and ultimately any output of AI model workloads is delivered over a private network or the public Internet.

3. Storj performs much better than traditional storage like AWS S3, especially in terms of global distribution. Storj was used to store the LAION-5B dataset's English-only embedding and it was found to perform significantly better than AWS S3.

Try it for yourself:
https://link.storjshare.io/s/jvdbhbogkj5xvd4occnk5c5j3nya/datasets/laion5b/embeddings/laion2B-en/

4. The StarCoder AI model from HuggingFace was downloaded faster from Storj than from the traditional AWS S3 bucket, showing that Storj can outperform traditional methods of data storage and retrieval.
https://github.com/storj/huggingface-hub-storj-patch

5. Storj is not only useful for AI training and model distribution, but also for high-performance computing data storage and scientific data sharing. This versatility was demonstrated by an experiment conducted by Prof. Antonin Portelli at the University of Edinburgh.
https://www.storj.io/resource/university-of-edinburgh-performance-report

6. The Storj network adapts well to concurrency and parallelism, providing a built-in performance boost as data is scattered worldwide. Storj offers similar performance to S3 Transfer Acceleration, but at much lower costs. This makes it a cost-effective solution for distributing large amounts of data globally.
https://www.storj.io/blog/new-test-of-storj-performance-by-univ-of-edinburgh-shows-2x-improvement

7. Storj can be easily integrated into existing applications with native language bindings or hosted gateway solutions, without requiring changes in application behavior.
https://docs.storj.io/dcs/getting-started/gateway-mt
https://docs.storj.io/dcs/getting-started/quickstart-uplink-cli/prerequisites

8. This study concluded that Storj is an excellent solution for global file distribution, particularly for AI training data and models, due to its high speeds, low costs, and easy integration. It was suggested that other organizations might benefit from adopting Storj to improve their own model distribution strategy.

STORJ

Storj
1450 W. Peachtree St. NW #200
PMB 75268
Atlanta, GA 30309-2955
info@storj.io