EXHIBIT 73

(FILED UNDER SEAL)

Dawn Drain Aug 15, 2023

Deduplication
- Algorithms
  - Fuzzy, file-level dedupe
  - Fuzzy, "paragraph"-level dedupe
  - Substring dedupe aka exact dedupe
  - Hashy aka hash-happy dedupe
- Scans
  - Basic signs of life scans
  - Optimal dedupe parameters
    - Deduping cc.p8 graphs
    - Deduping code graphs
  - Moe
  - Teal-like distribution
  - Multi epoch results
- How much stuff gets removed
  - Per dataset
- How to
- Deduping multimodal data

Memorization
- Large LMs memorize A LOT, like A LOT
- How many repetitions before a string is memorized?
- How many unique things are memorized?
- What's getting memorized?
  - cc.p8 samples
  - Code samples
- How harmful is memorization?
  - Danny's paper
  - More Synthetic Dupes Experiments
- Forgetting
- Dropout
- Estimating the CM hit for tem/tea[redacted]
- Memorization is[redacted]
- Memorization is[redacted]
- How much capacity does it take to memorize
- Mechanistic explanation of memorization

Todo's

Also see Kipply's doc Exact Dedupe



Deduplication

Algorithms

Fuzzy, file-level dedupe

Ben implemented our standard file-level fuzzy dedupe algorithm, which he also used for the GPT-3 paper.
The goal is to treat two files as (fuzzy) duplicates if their bag of words has high jaccard similarity (aka set similarity). To accomplish this, we use a family of hash functions (10 hash functions in practice). For a given hash function, we hash each word in the file, and sort according to those hashes, and then take the minimum word. The probability that two files have the same minimum word is equal to their jaccard similarity! Kind of cool. So if two files have the same minimum word for all ten hashes, then they're probably very duplicated.

We can hash the tuple of the ten minimum words to get a "rehash", which then let's us dedupe in linear time (or nlogn, whatever)

As a small optimization we just consider the first [redacted] of each file.

Fuzzy, "paragraph"-level dedupe

Rather than be stuck working with whole files, we can also fuzzily dedupe at the "paragraph" level. I put "paragraph" in quotes since what we're actually doing here is targeting a length of say 500 tokens, and then defining the start of a new paragraph as when the tuple of the previous five tokens hashes to 0 mod 500.
(Idk if we're going to be able to get this in in time for teal :/. Pranay and I paired on looking at what this was removing, and it was mostly removing good stuff like shuffled file headers or filled-in templates (e.g. a bunch for every different college saying "If you want to go to [College X], then you should talk to your guidance counselor..." etc.))

Substring dedupe aka exact dedupe

The goal here is to remove substrings that appear many times in the corpus e.g. the GNU license probably shows up like 100k-1m times.

For concreteness, let's say we want to remove all but the first occurrence of any string of >100 chars that shows up >20 times.

The algorithm works by first constructing a giant suffix array for the dataset (or a shard of the dataset for something like cc.p8 T.T). By definition, a suffix array is a sorted list of all suffixes of a string (in our case the string is the whole dataset). (Naively this is a quadratic data structure, so in practice we just hold 5-byte pointers to where the suffixes start).

Given a suffix array and a query substring, we can do a binary search to find the first occurrence of that substring, and then find the last occurrence by either walking along linearly or doing a second binary search. So this lets us count how many times any given substring appears in the dataset.

We build on this code base https://github.com/google-research/deduplicate-text-datasets/tree/master
Our code is in `data_pipeline/data_pipeline/deduplicate_text_datasets`. It uses rust, lots of threading, and takes advantage of the i4i.metals' big 32TB disks.

Hashy aka hash-happy dedupe

This works for both substring/exact-dedupe as well as fuzzy dedupe, although so far we've almost always used it for substring/exact-dedupe

We take a 3 phase approach (with also a phase 1b and 2b...)

1. In phase 1, compute hashes for all ngrams, and bin them together based on their modulus (also be sure to note where the ngram came from!)
(for now I'm basically splitting on whitespace to get the ngrams)
2. In phase 2, compute the Counter of how often each hash appears, because we can.
More importantly for phase 2, bin *again*. This time we write the locations of the ngrams to remove, and bin by s3_path hash
3. Finally, actually remove those stretches from each s3_path that we started with

This has the advantage of scaling to *all* our data, unlike the suffix array approach

Scans

Basic signs of life scans

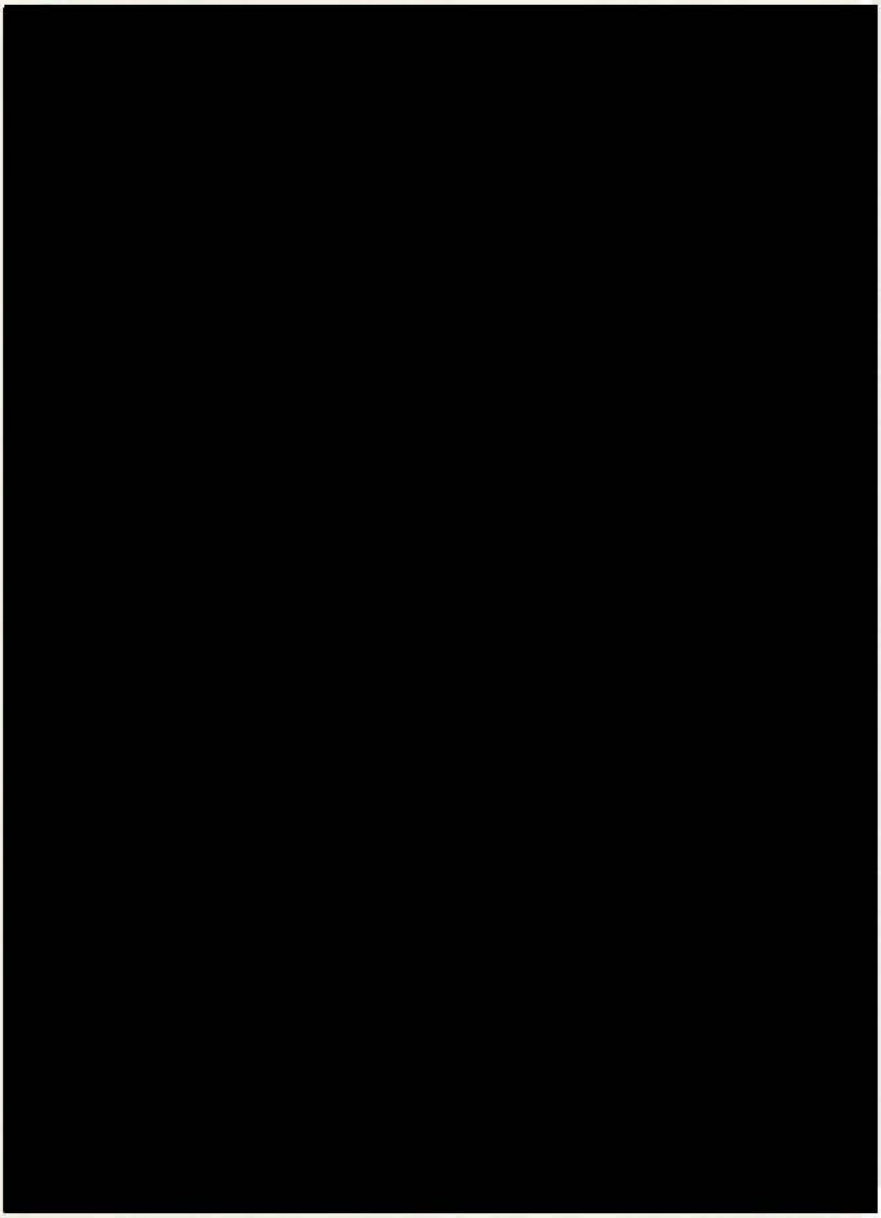
The first scan I tried was running substring dedupe on cc.p8 for an [redacted]
The results were pretty awesome!


compute multiplier
fraction_ir


ute multiplier

https://anthropic.slack.com/[redacted]

#+ Also sent to the channel



11 13

This was removing all but the first occurrence of any string of [redacted] appeared [redacted] times in this 128GB shard [redacted]

The next followup was deduping code. I tried a [redacted] here. Looks like we get a [redacted] on the monorepo eval, independent of scale, okay, and a [redacted] on natural language
(although note that those [redacted] are still with the improved exponent from deduping CC)
https://anthropic.slack.com/[redacted]

[redacted]



Optimal dedupe parameters

The original substring dedupe paper recommended using length_threshold=100 and freq_threshold=1. Their code also removes "all" copies of any duplicated string! So very aggressive dedupe!
So far our results do point to [redacted] although there's a tradeoff from [redacted]
Note also that when I'm using [redacted] that's kind of like a [redacted] since right now we can't [redacted] deduping our giant datasets in their entirety. (Although it's also kind of just like a [redacted] in that we're only training on that one shard...)

results for tweaking the freq_threshold for the [redacted] mix (just deduping the cc here)
Looks like lowering the freq_threshold [redacted] is somewhere between [redacted] and conversely increasing it to [redacted] is consistently like a [redacted]

Deduping cc.p8 graphs


Compute Multiplier
13:35:28 spectra.analysis.fitting:130 822587 [00000] Extrapolating CM; extend your baseline for more accurate results
Compute Multiplier



13:35:28 spectra.analysis.fitting:130 822587 [00000] Extrapolating CN₂ extend your baseline for more accurate results





13:35:28 spectra.analysis.fitting:130 822587 [00080] Extrapolating CM; extend your baseline for more accurate results



Deduping code graphs

Moe

Bryan said that moe gets [redacted]

Teal-like distribution

Since commoncrawl and code are especially duplicated, you'd think these results would diminish when looking at a more teal-like data distribution. Devi did a scan on some substring-deduped teal data, and again saw great [redacted]

[redacted]

```
07:56:01,923 spectra.analysis.fitting:130 689352 [00000] Extrapolating CM; extend your
07:56:01,926 spectra.analysis.fitting:130 689352 [00000] Extrapolating CM; extend your
07:56:01,928 spectra.analysis.fitting:130 689352 [00000] Extrapolating CM; extend your
07:56:01,930 spectra.analysis.fitting:130 689352 [00000] Extrapolating CM; extend your
07:56:01,932 spectra.analysis.fitting:130 689352 [00000] Extrapolating CM; extend your
07:56:01,935 spectra.analysis.fitting:130 689352 [00000] Extrapolating CM; extend your
```



[redacted]

[redacted]

```
07:56:01,958 spectra.analysis.fitting:130 689352 [00000] Extrapolating CM; extend your
07:56:01,960 spectra.analysis.fitting:130 689352 [00000] Extrapolating CM; extend your
07:56:01,961 spectra.analysis.fitting:130 689352 [00000] Extrapolating CM; extend your
07:56:01,963 spectra.analysis.fitting:130 689352 [00000] Extrapolating CM; extend your
07:56:01,964 spectra.analysis.fitting:130 689352 [00000] Extrapolating CM; extend your
```

[redacted]

Compute Multiplier

compu

[redacted] but still very good!

[redacted]

Multi epoch results

Devi has shown that the [redacted] is basically [redacted] whether or not we do exact dedupe. Her notebook is at devi/exact%20dedup%20vs%20vine.ipynb

Eyeballing, looks like [redacted] hit increases by [redacted]. So e.g. filtering out [redacted] is like training for [redacted], which is like going from [redacted] or something, which is like a [redacted] Okay, so I guess we're happy to [redacted]


compute multiplier



Here's the same graph reformatted to show [redacted] on the y-axis (so yeah, [redacted] In the most pessimistic case (which we [redacted] having less data means you [redacted] In that case an extra [redacted]

(from /notebooks/dawn/code_data/cc_filtering/memorization/effect_of_repeated_epochs.ipynb)

:

[redacted]

cm

:

[redacted]

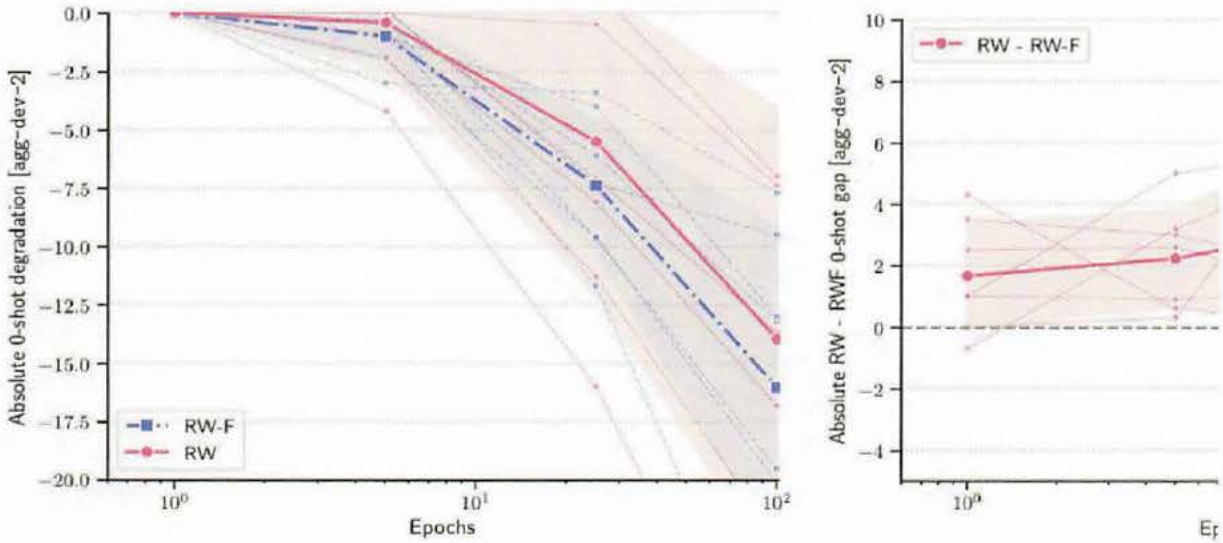
(note that this is without recalibration or dropout)

The refined web paper claimed that dedupe decreased the penalty from repeated epochs, although they presented their results in a confusing way T.T [redacted] (also note that their baseline doesn't even have "any" dedupe, [redacted]

E.3. Does deduplication help with multiple epochs?

Earlier in this work, we outlined that to scale pretraining data, practitioners had two choi
which is the avenue we chose to pursue; (2) train models on multiple epochs of the same d
in the ability of larger models to sustain multiple epochs without adverse effects (Hernande
A fairly rational question regarding (2) is whether deduplication may improve the situatio
may be able to sustain more epochs without compromising model quality.

We train 1B parameters models on 30GT of RW and RW-Filtered. We keep the number of
for 1, 5, 25, and 100 epochs. This is a small-scale, limited set-up, which would have to
results. We plot the degradation in performance compared to a single epoch in Figure 7(
RW-F in Figure 7(b). We find that the absolute degradation is less important for RefinedW
furthermore, the gap widens with increasing number of epochs. However, we observe sig

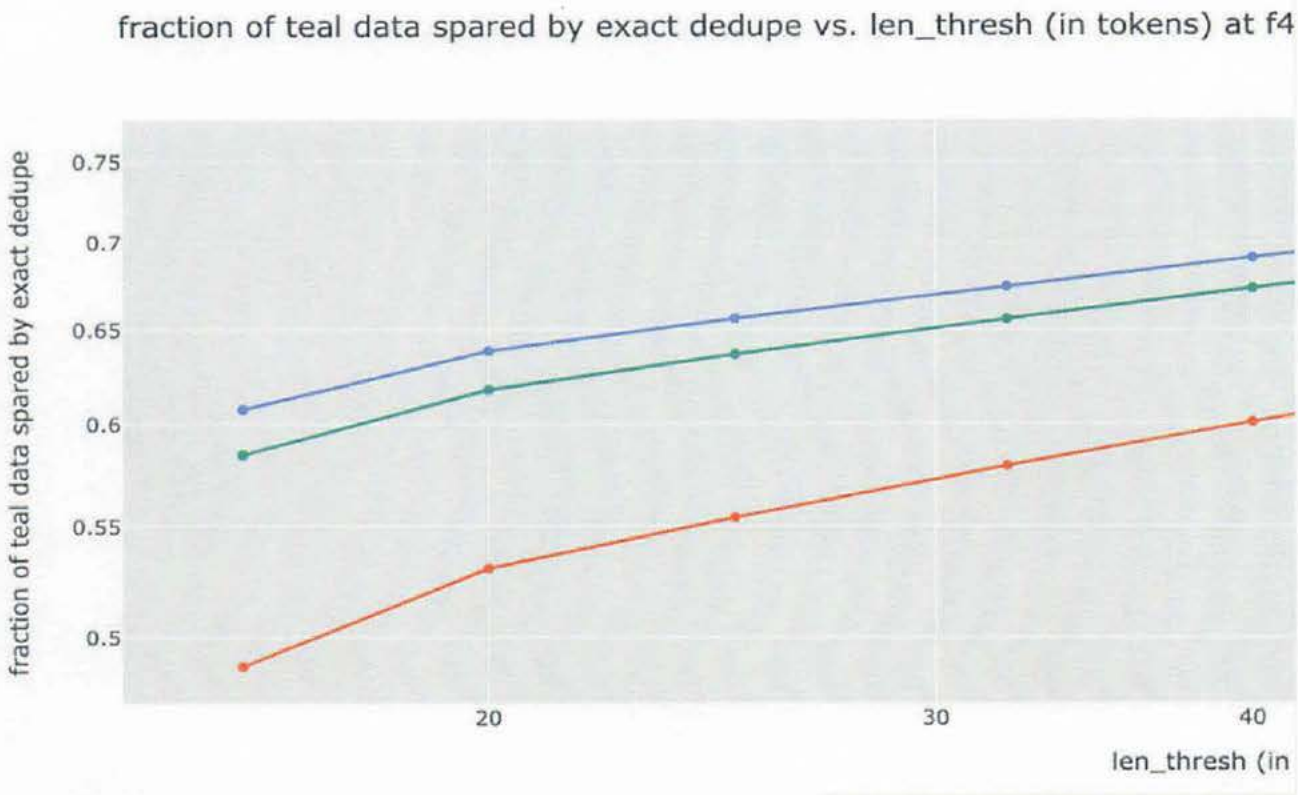


(a) Degradation compared to 1 epoch (b) Gap between

Figure 7. **Deduplication may reduce the degradation in performance incurred by multiple epoc**
only performed at small-scale (1B models trained on 30GT), and we see high variability in outcomes
measured on the `agg-dev-2` aggregate (HellaSwag, PIQA, ARC, BoolQ, COPA, MRPC, SciQ). I
and 1-σ standard deviation across all tasks in the aggregate in transparent.

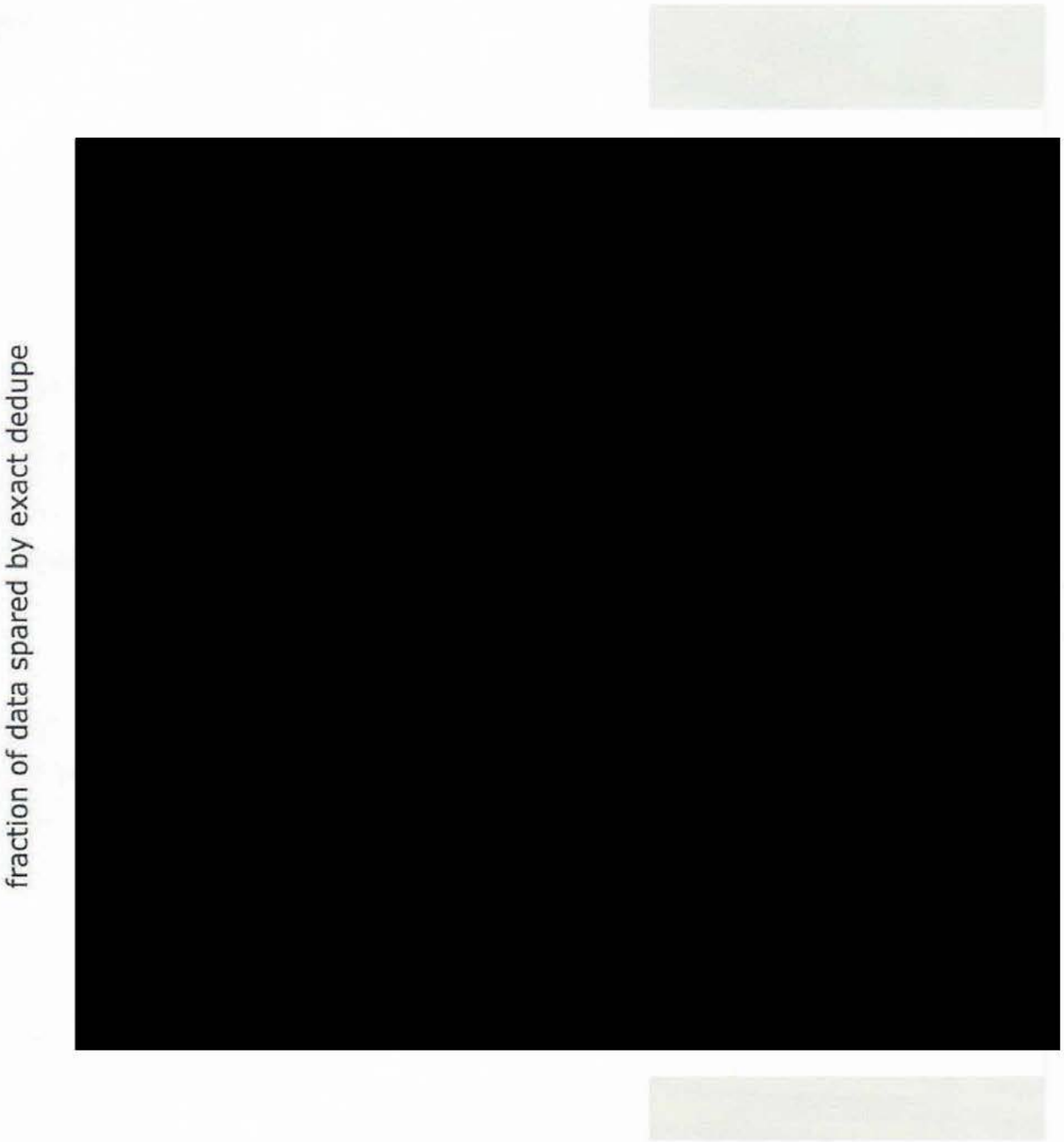
How much stuff gets removed

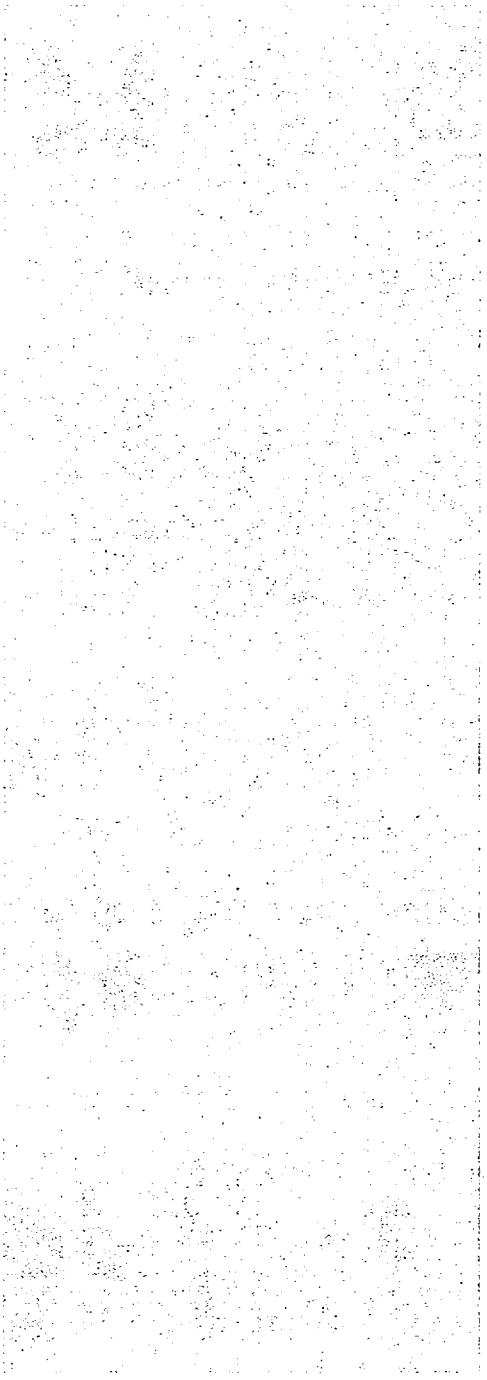
I deduped *all* our teal data using [redacted]
tokens. Those [redacted] settings would remove approximately [redacted] of our data 😱
(this is kind of a biased sample, e.g. we wouldn't [redacted] even if we removed [redacted]
[redacted] since we're only [redacted]



If you disaggregate per data-mix component, looks like we keep most of the really good stuff like (90% of books and papers), gut half(!) of code, and like a third of commoncrawl (this is with excising duplicated ngrams + dropping docs where 70% of tokens would be excised)

https://anthropic.slack.com/archives/C05JY8HCVT2/p1695408985636069


fraction of data spared by exact dedupe



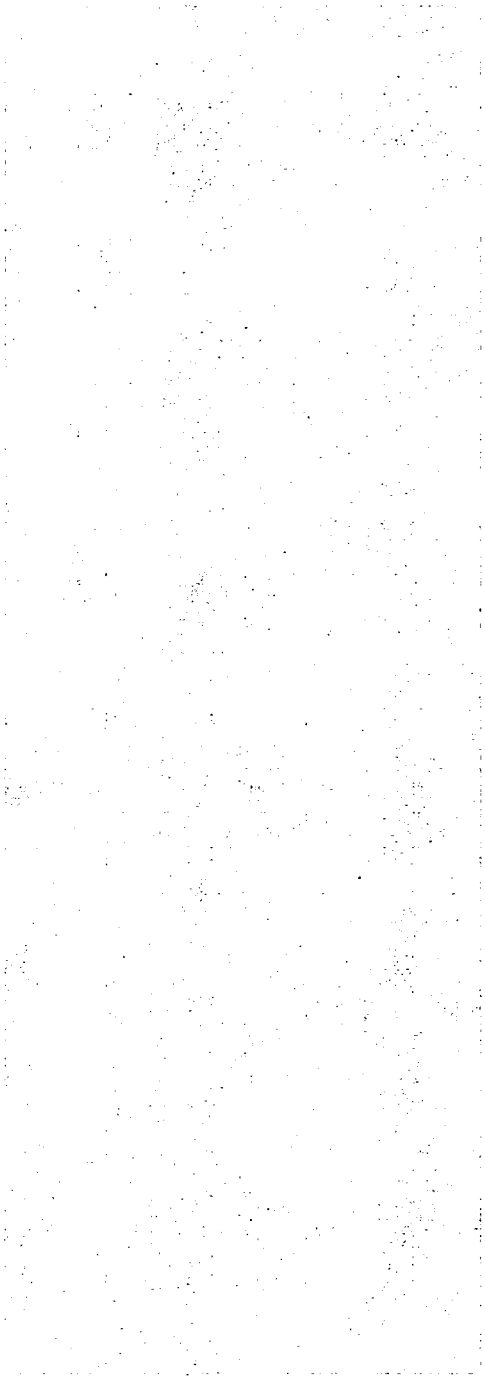

fraction of data spared by exact dedupe




fraction of data spared by exact dedupe



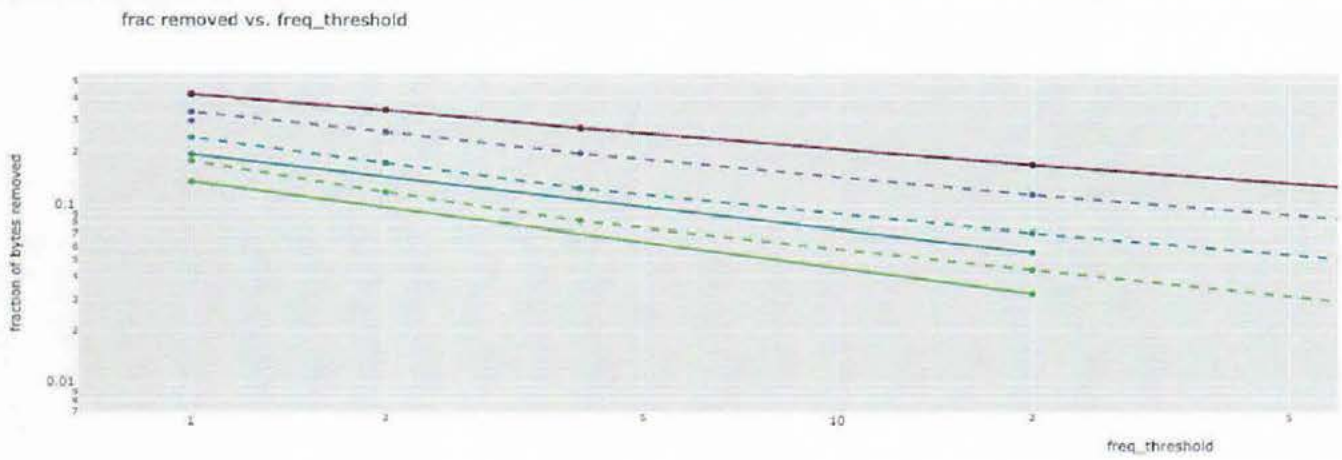

fraction of data spared by exact dedupe
len_




fraction of data spared by exact dedupe

(older graphs below)

Here are results varying freq_threshold and length_threshold for a single ~100GB shard



I also have some older results using freq_threshold=1
(sorry that the graphs are kind of hard to read)

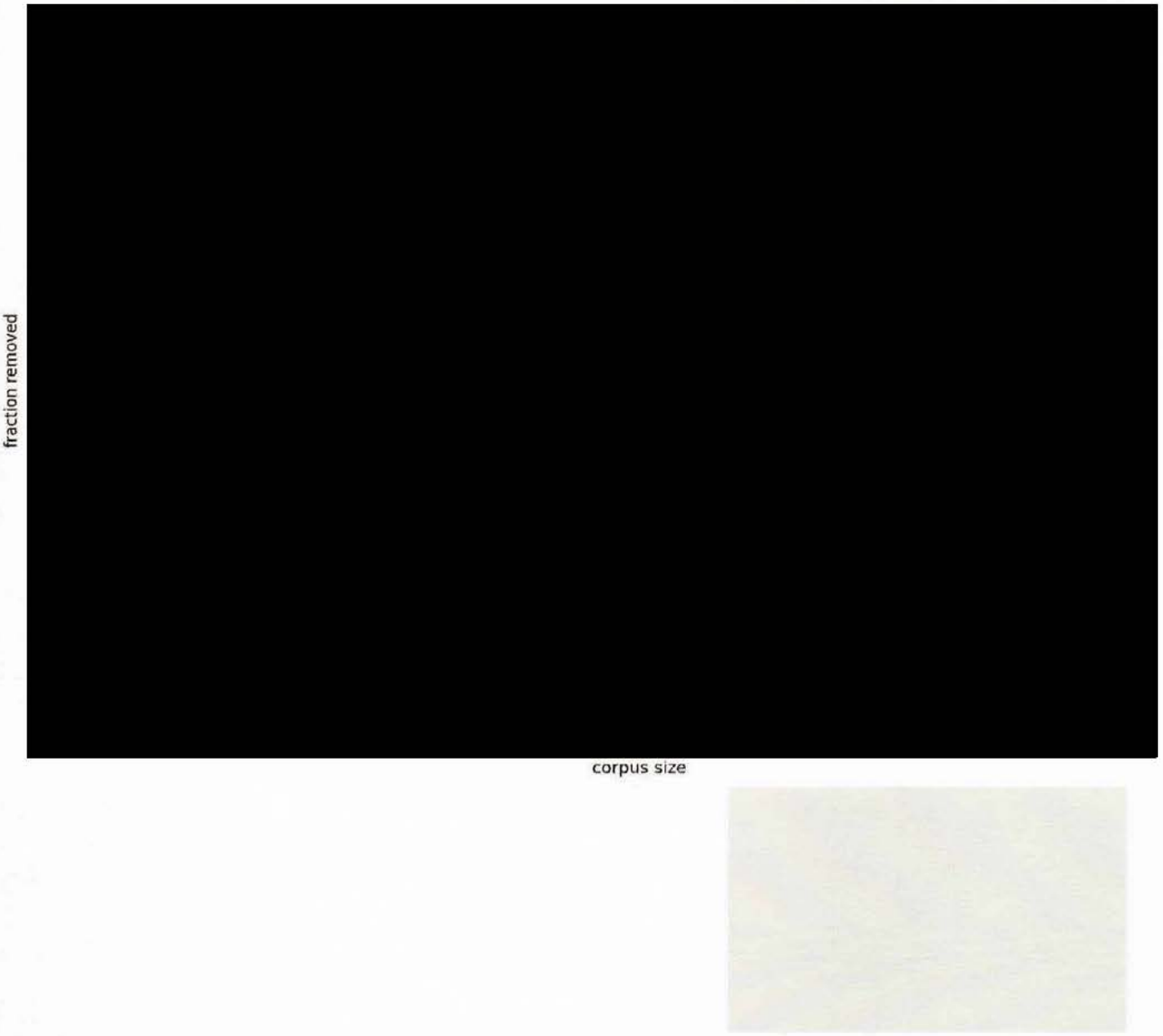
Per dataset

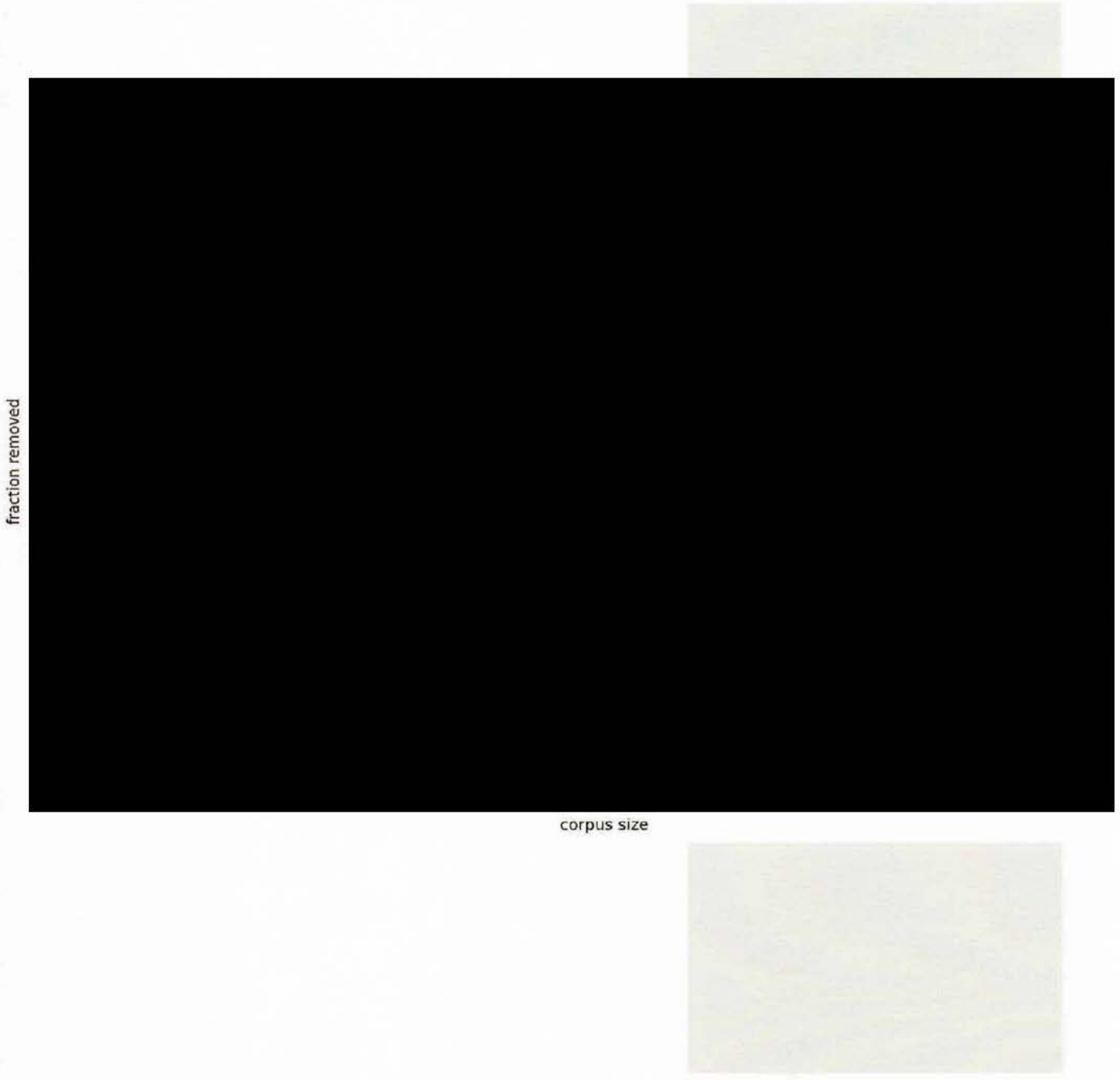
These graphs are all looking at freq_threshold=1 and varying the length threshold, on at most 64GB of data for cc (so like 3% of cc.p8) and 128GB of code (so like 10% of that dataset). At these scales, length_threshold=100 removes 20-30% of cc, and *half* of code lol

Law is also pretty duplicated, eh. Papers, pdfs, and books are much less duplicated (although pile of books and cc.pdfs still removes 10% at freq_threshold=1, eh)

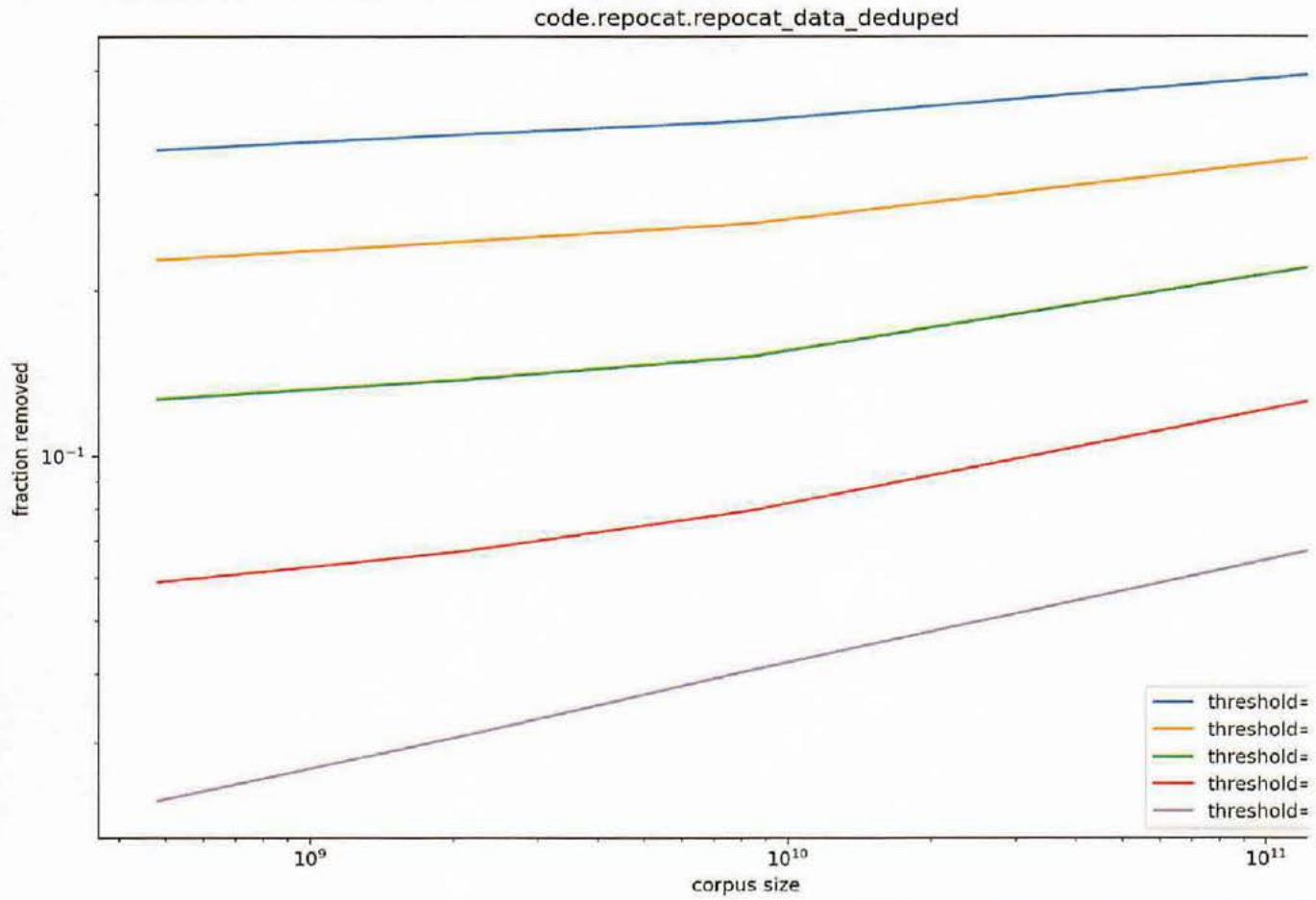
Commented [1]: In my above scan on teal-like distribution, I found that including/excluding the cc.pdf datasets in exact dedup didn't affect the CMs [Author:Devi Borg],[Creator:1297307460],[Date:8/21/2023 2:38:00 PM]

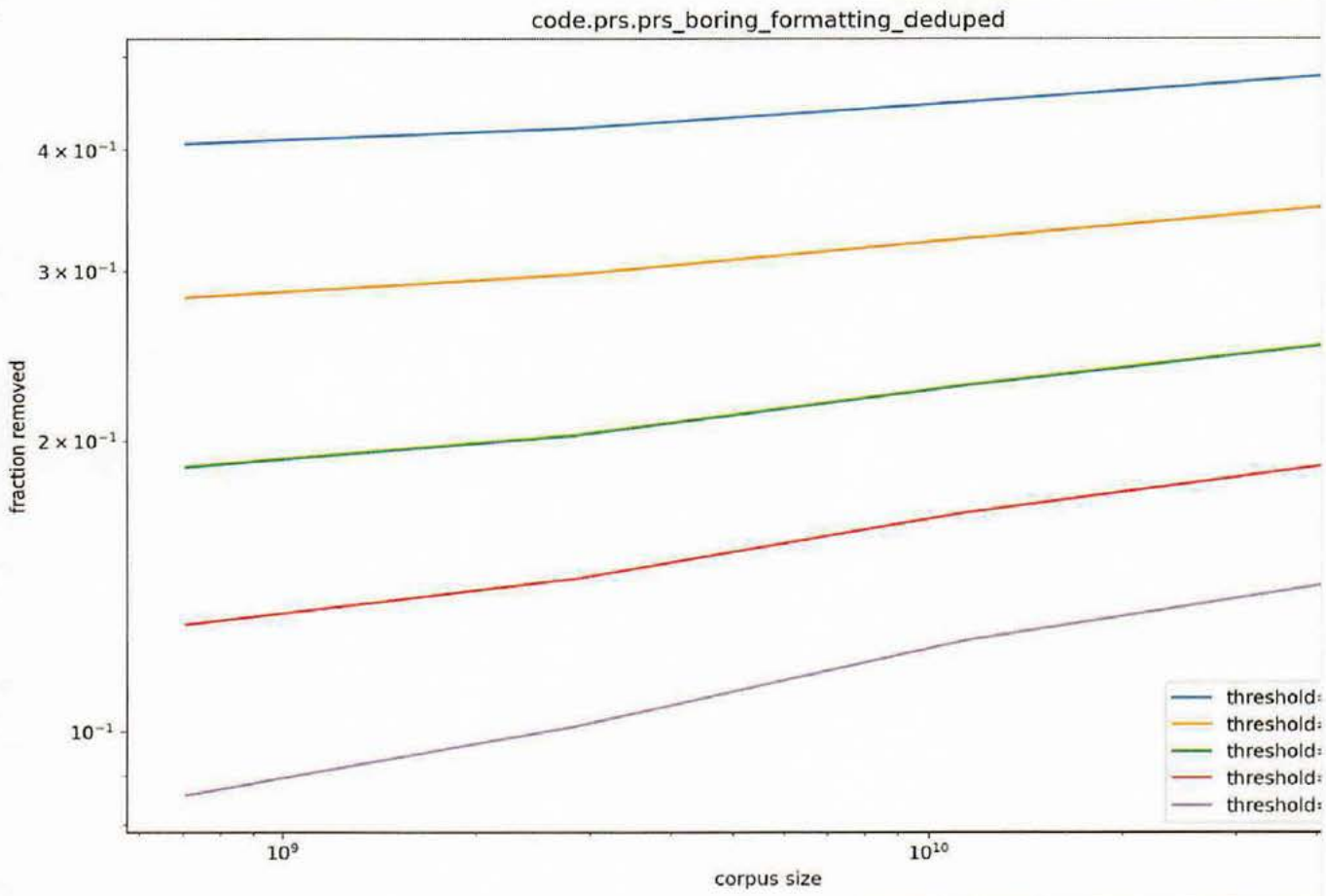

fraction removed
corpus size


fraction removed
corpus size

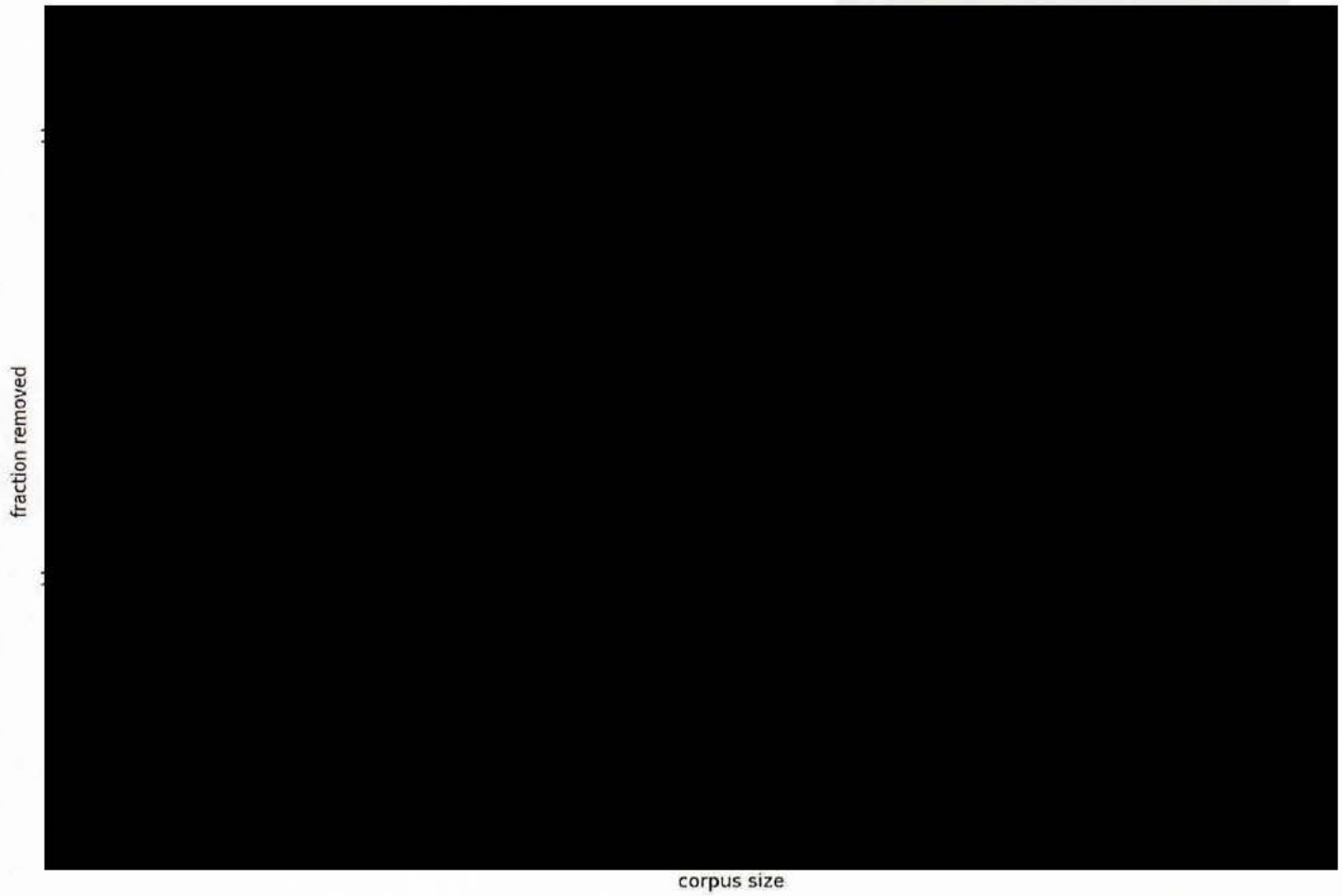

fraction removed
corpus size

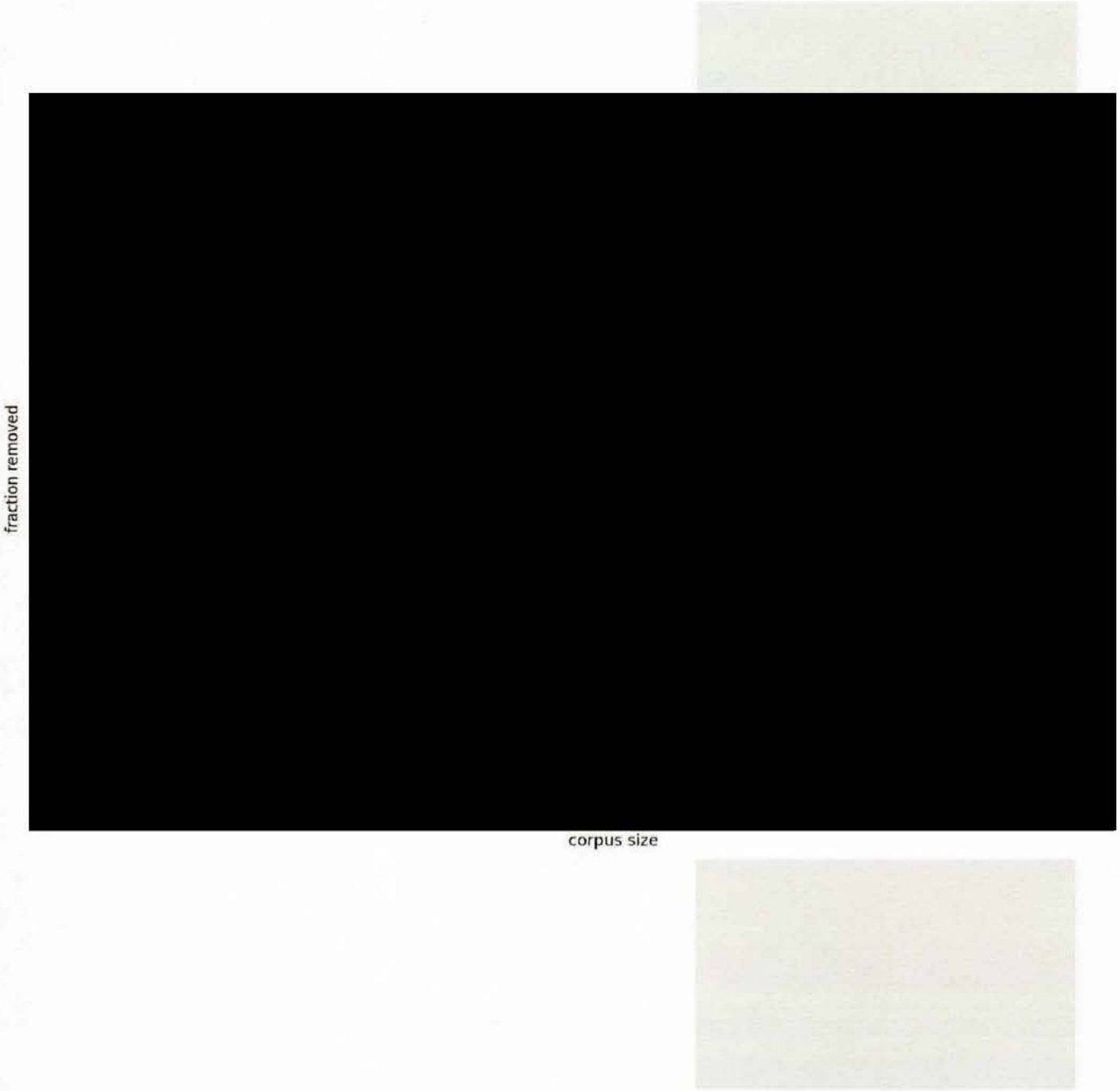
Code is very duplicated
Even with a tiny corpus size, prs are still super duplicated when using freq_threshold=1

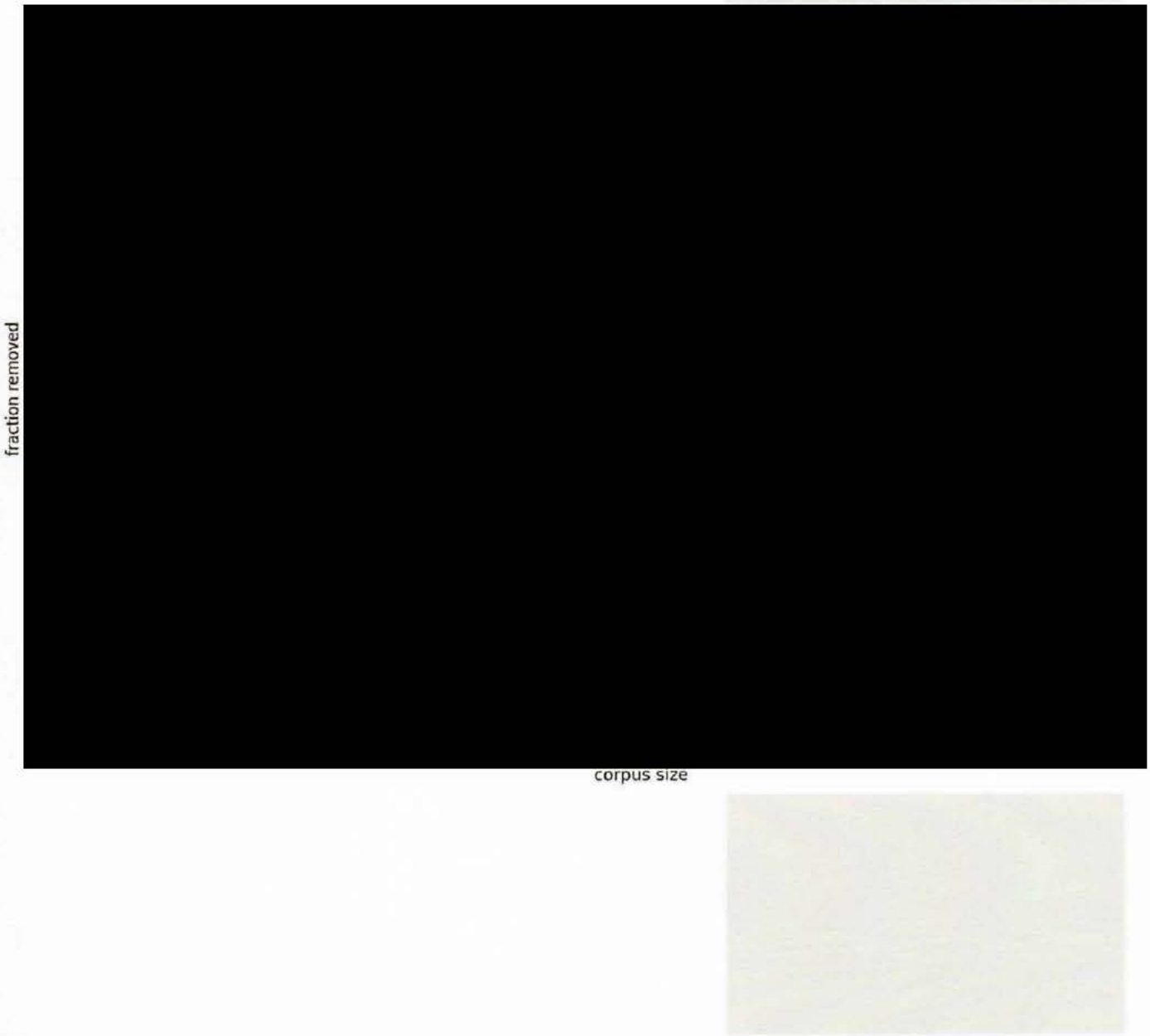


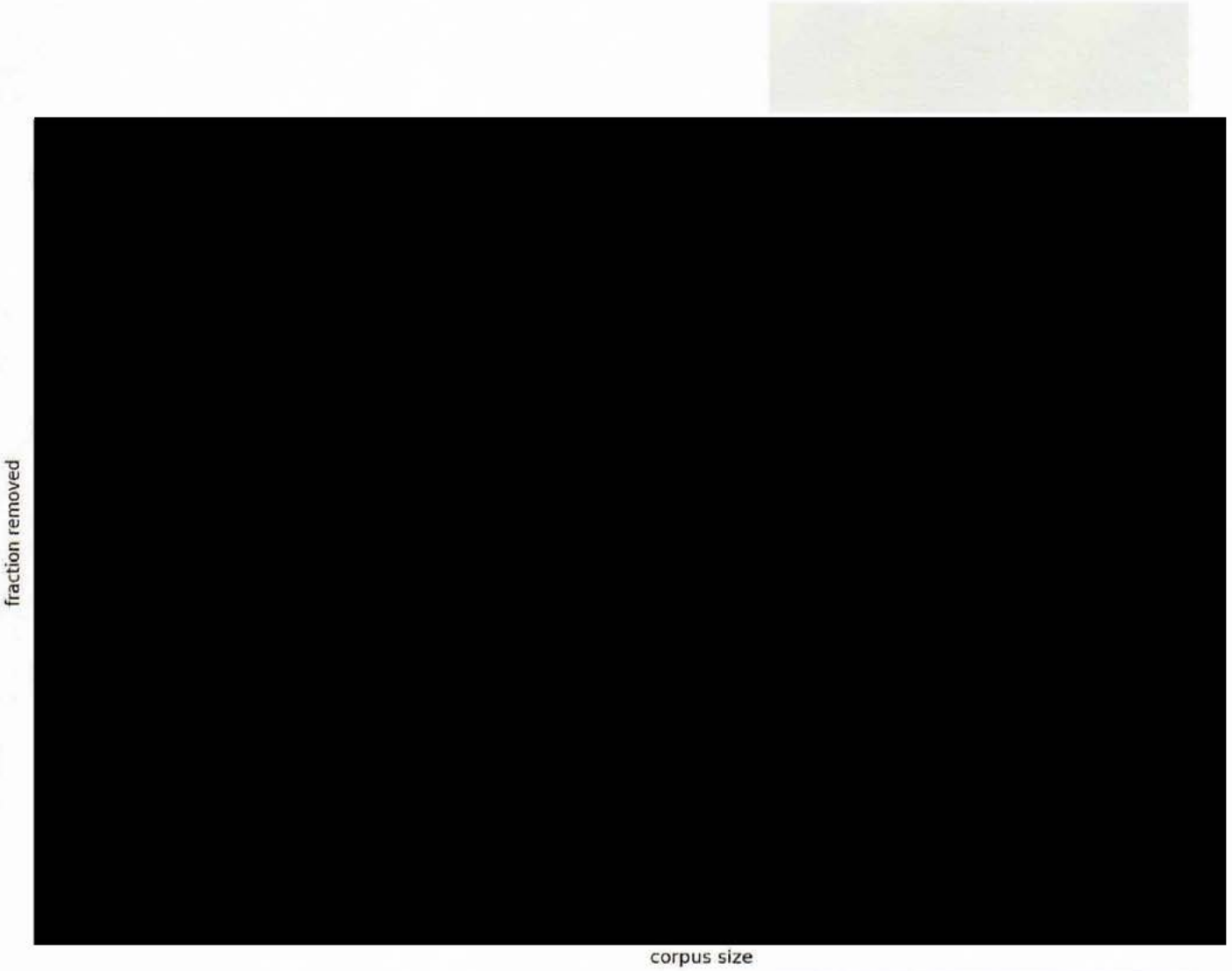


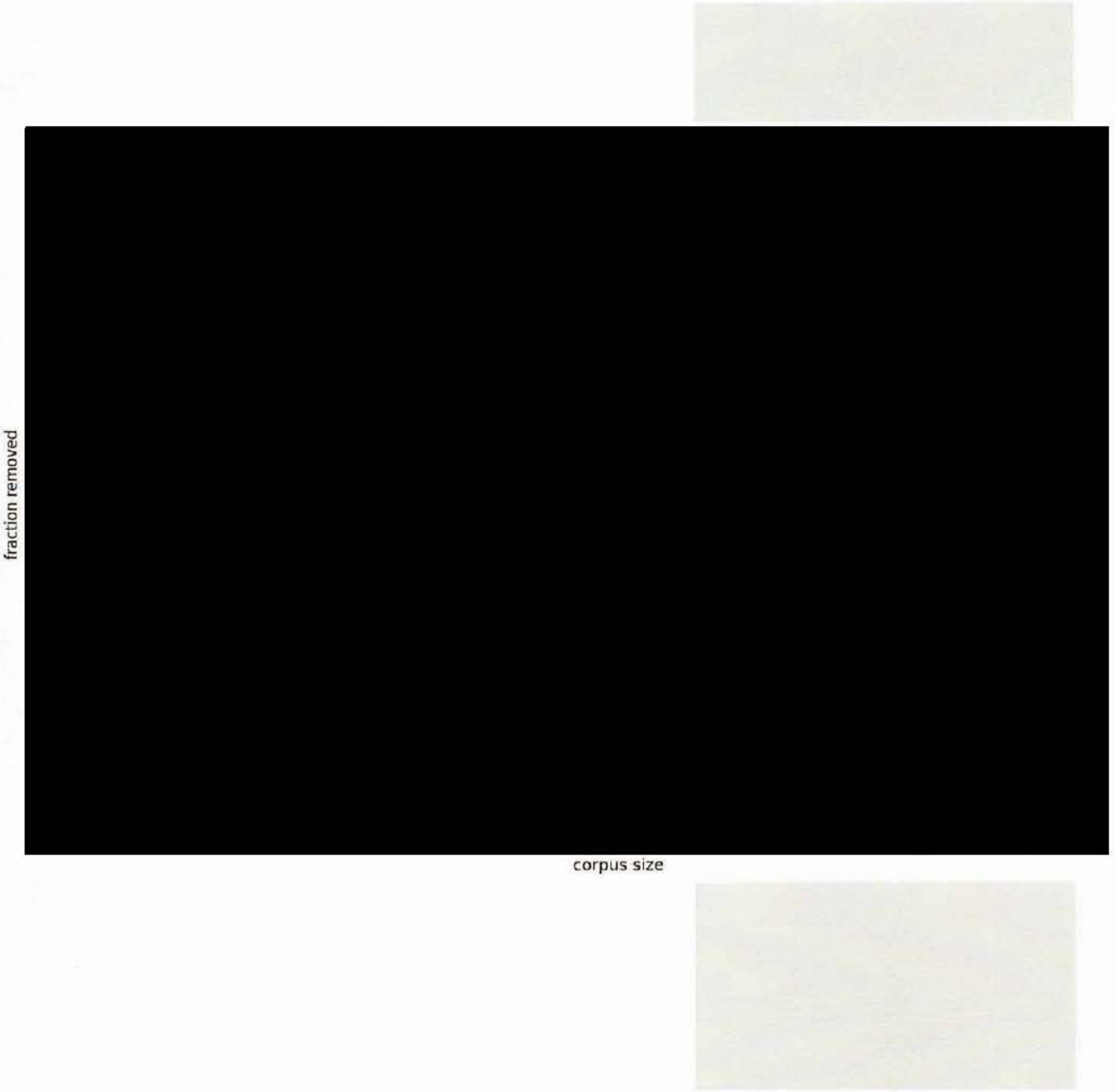
Pdfs, books, and papers are a lot less duplicated


fraction removed
corpus size


fraction removed
corpus size


fraction removed
corpus size


fraction removed
corpus size


fraction removed
corpus size

How to

For the hash-happy substring dedupe and paragraph-level dedupe, I use sandbox/sandbox/dawn/cc_filtering/launch_hash_ngrams.py. Kipply also has a wrapper to integrate with data_pipeline

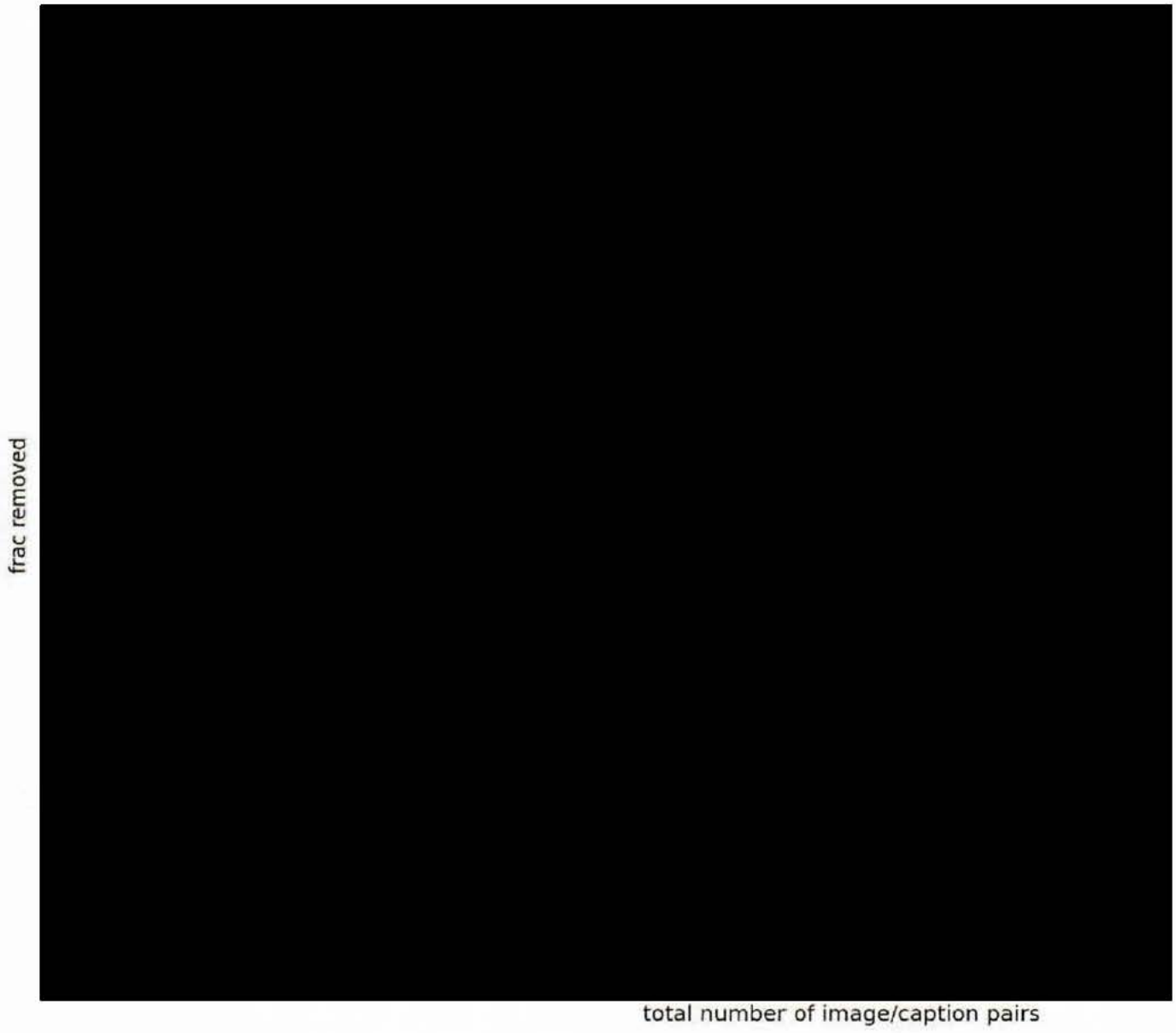
For substring dedupe using suffix arrays/rust, entrypoint is `data_pipeline/data_pipeline/deduplicate_text_datasets/src/run_pipeline.py`
(Note that this only scales to <1T uncompressed bytes)

We pretty much always use fuzzy, file-level dedupe by default.

Deduping multimodal data

pj and I have looked into deduping multimodal. [redacted]

Commented [2]: This seems right to me, but also [redacted] [Author:Nicholas Joseph],[Creator:1297307460],[Date:8/16/2023 5:24:00 AM]

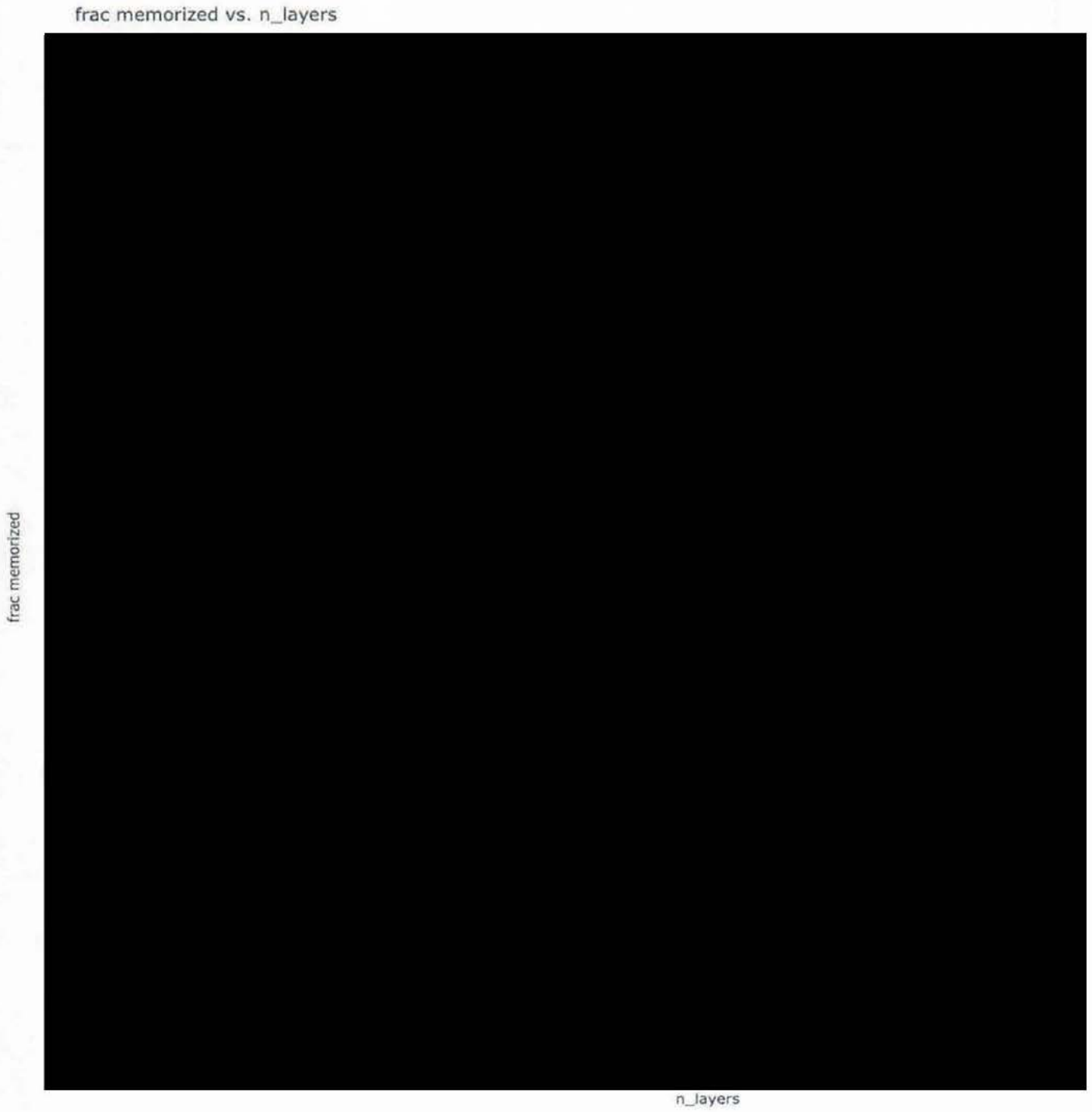

frac removed
total number of image/caption pairs

Also, 10-20% of images appear multiple times, and like half of those have *nearly* identical captions. (Also a solid chunk of those images are just placeholder images...). So we might want to do some kind of fuzzy dedupe on image captions for images that appear many times. See /notebooks/dawn/multimodal/image_similarity.ipynb

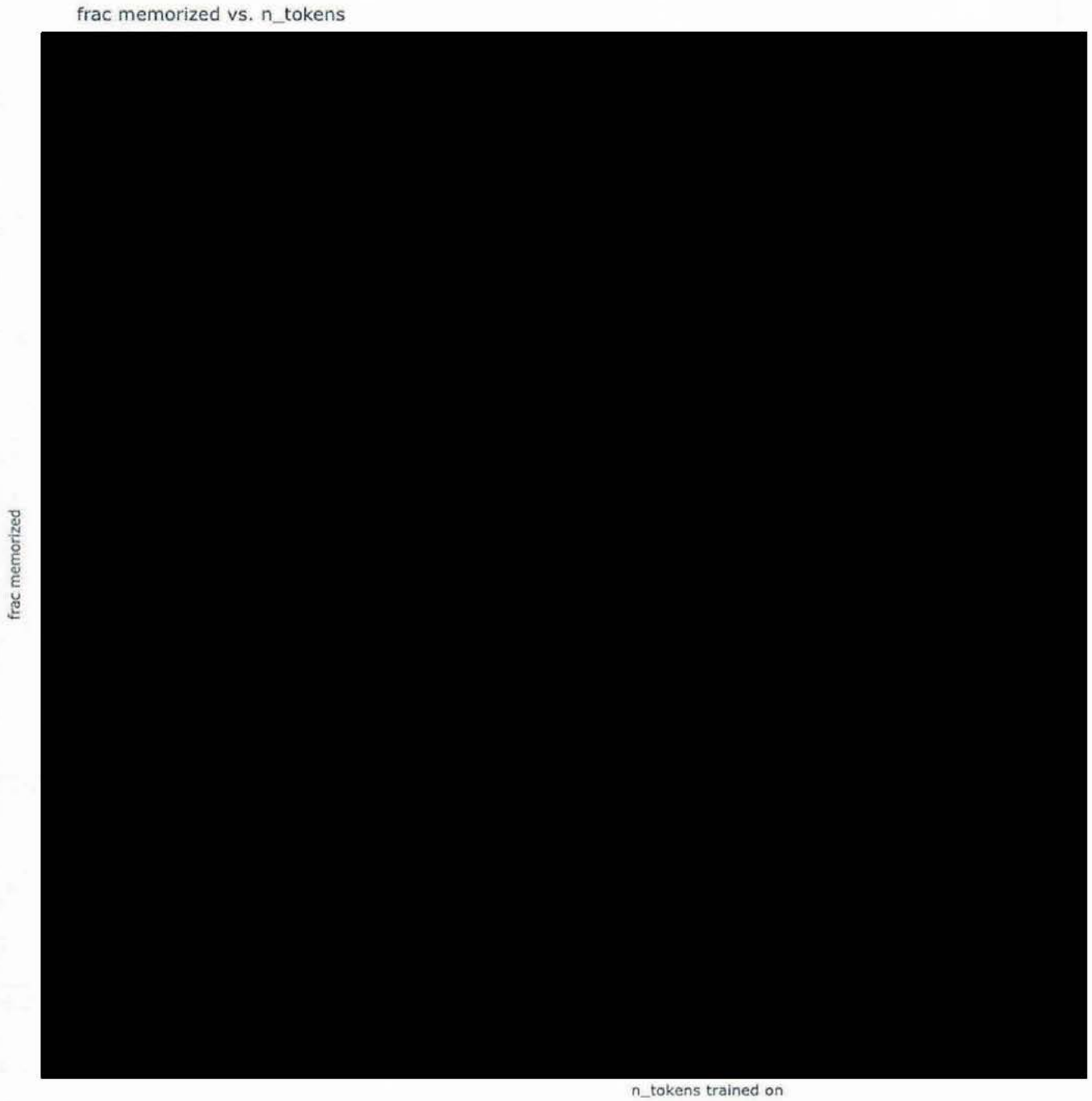
Memorization

Large LMs memorize A LOT, like A LOT

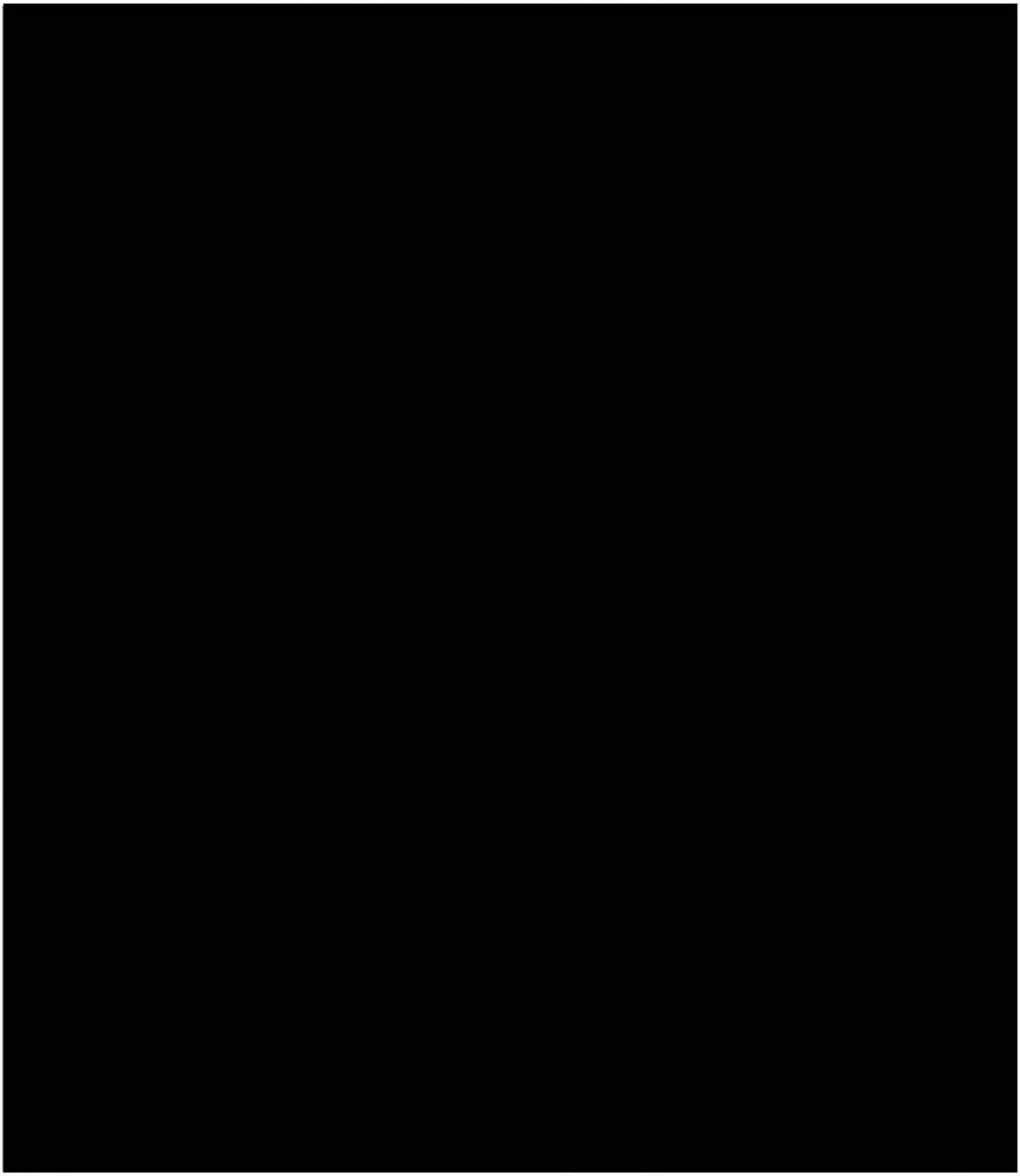
I looked at a few hundred thousand tokens from cc.p8, and tern had memorized a sixth of them down to a loss of...0.000001 (i.e. a confidence of 99.9999%)


frac memorized vs. n_layers
frac memorized
n_layers

Somewhat surprisingly the lines look [redacted]

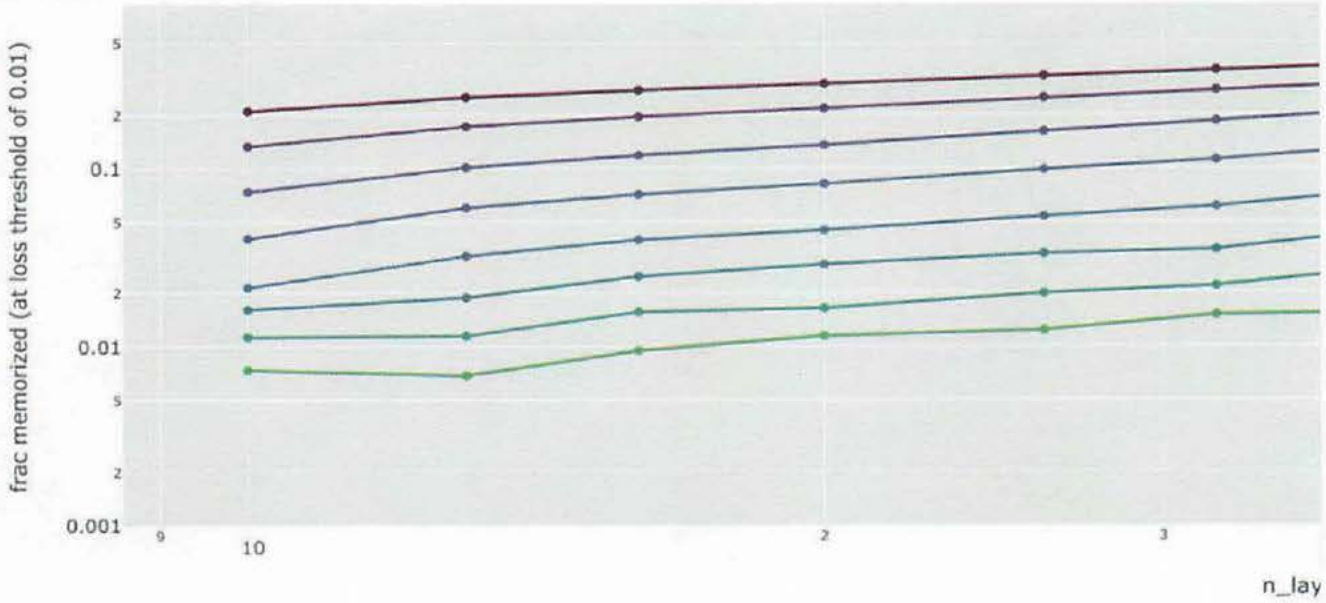

frac memorized vs. n_tokens
frac memorized
n_tokens trained on

Robert and Sam asked what fraction of those [redacted]


frac memorized (at loss threshold of 0.01)

https://s3-frontend.he.anthropic.com/anthropic-serve/plots/dawn/frac_cc_p8_memorized_stretches_vs_n_layers.html

Results are pretty similar for a sample of code



After looking at some samples, I found out that this sample of code is from Ben's original scrape, which contains forks, which are much more likely to be memorized, ok...

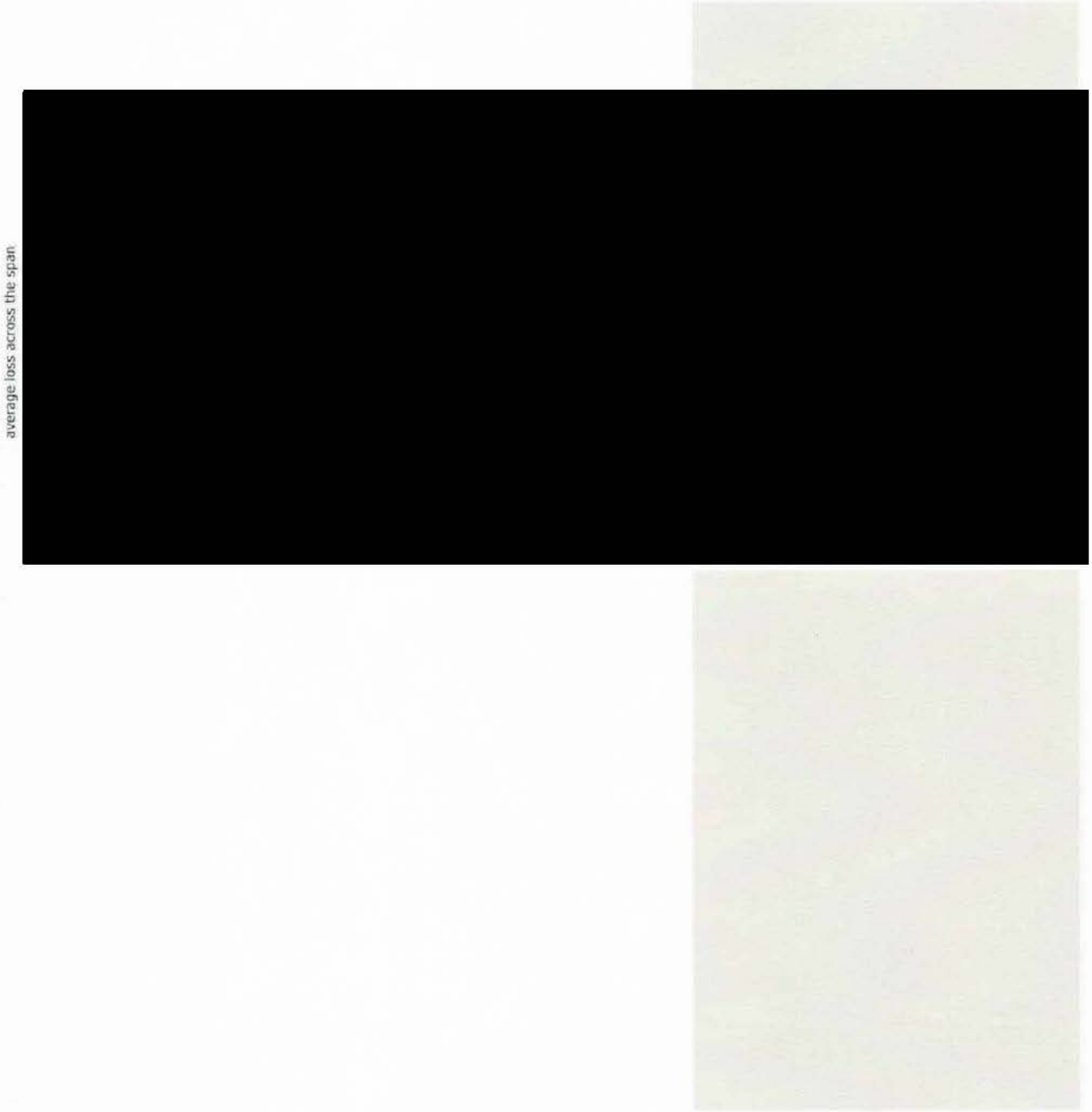
How many repetitions before a string is memorized?

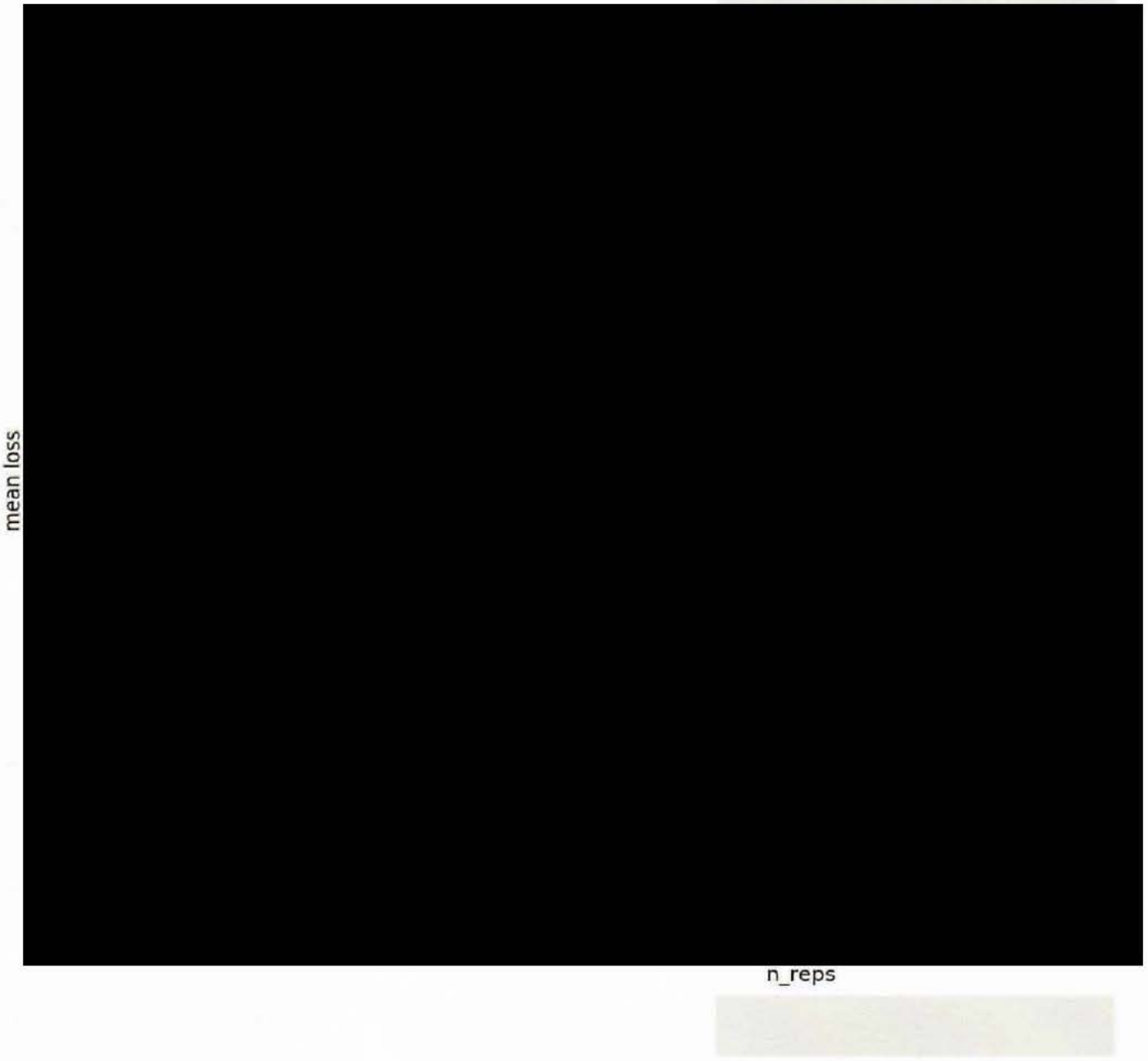
For my 4% sample of cc.p8, strings appearing 2-3 times have a 50% chance of being memorized, and strings appearing >10 times were pretty much all memorized. Naively

[redacted] It would be nice to get results with a bigger sample size, oh well

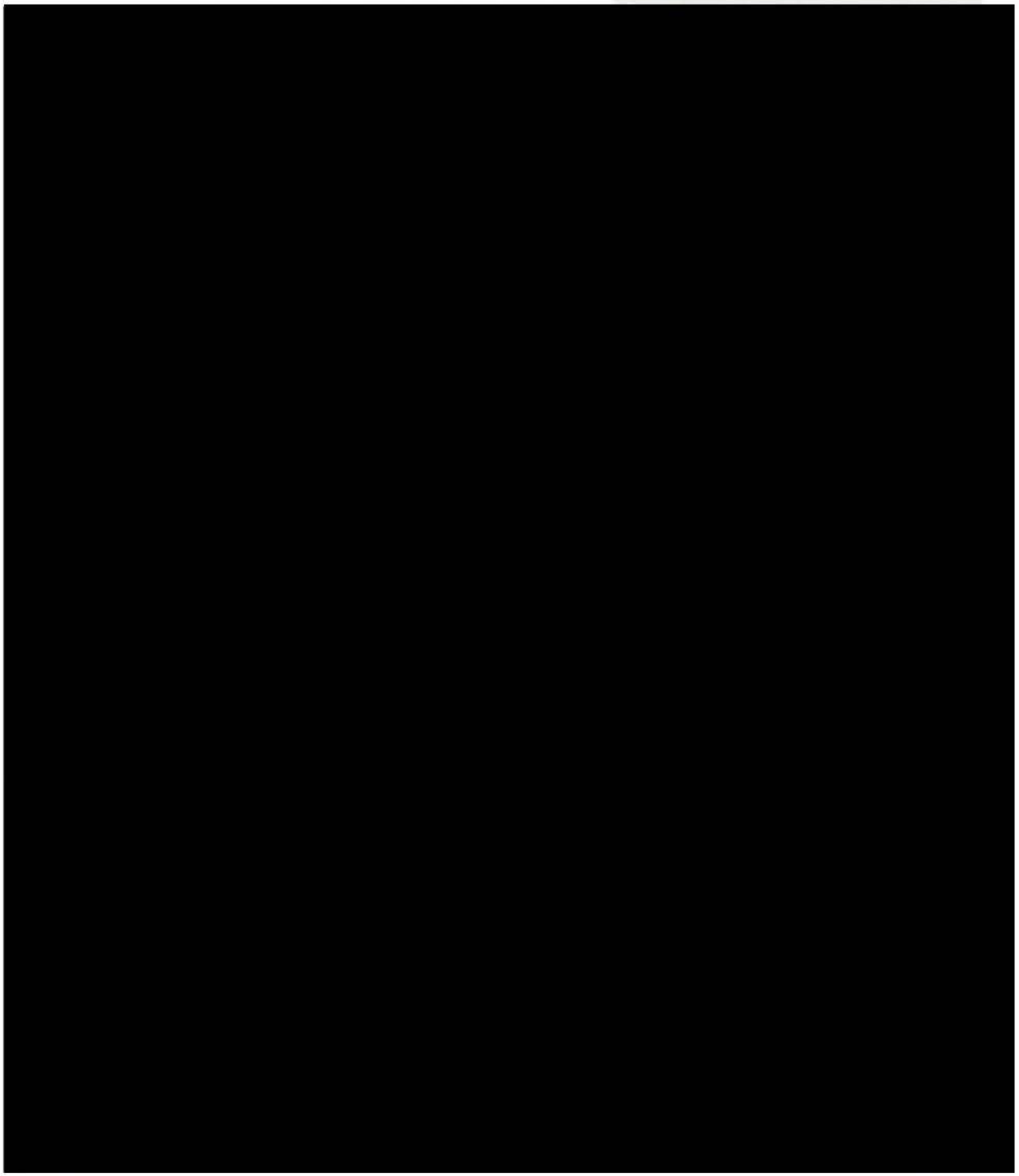


[redacted]

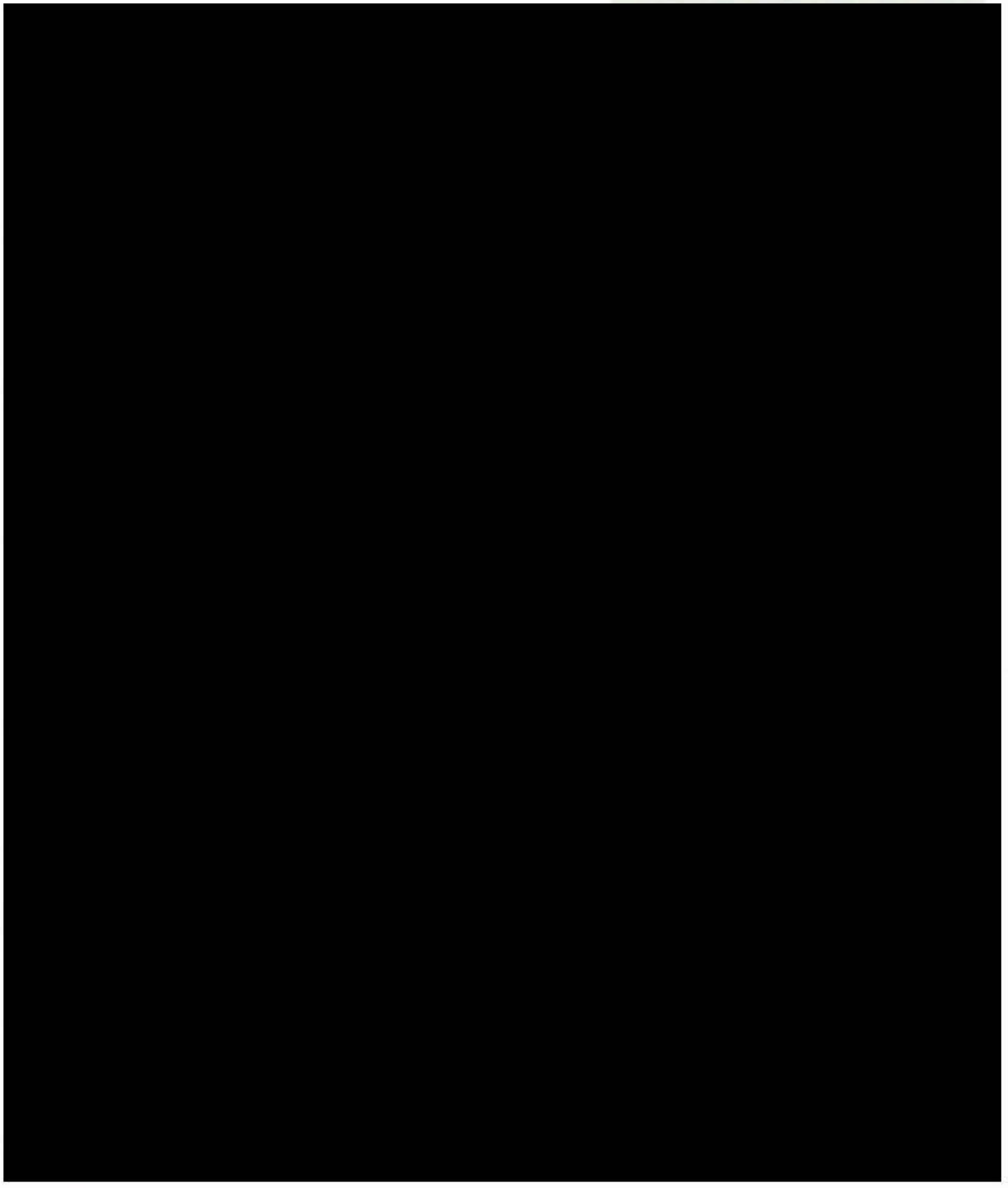

average loss across the span


mean loss
n_reps

Devi suggested [redacted]

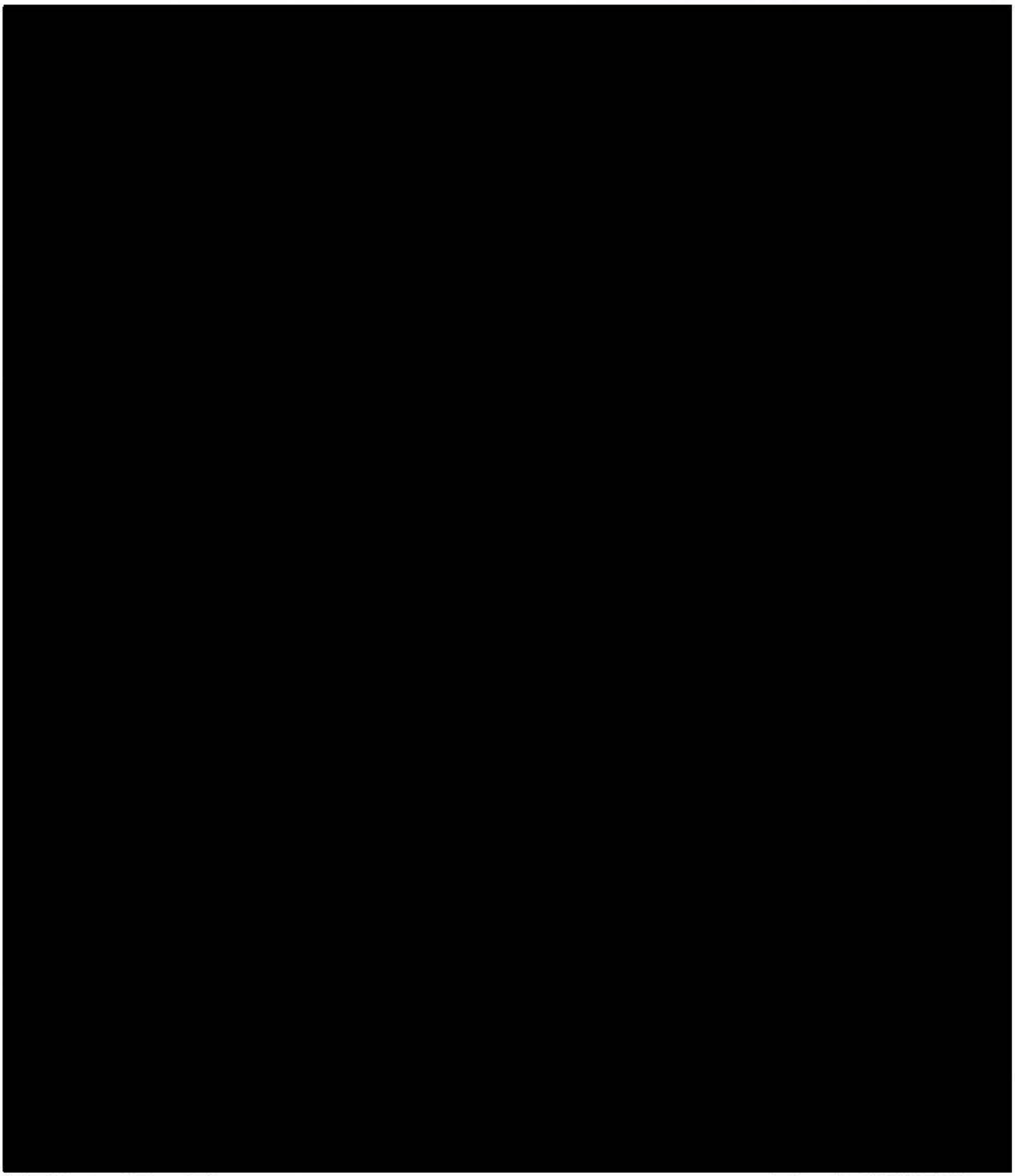

mean loss

The story's basically the same if you look at spans of [redacted]

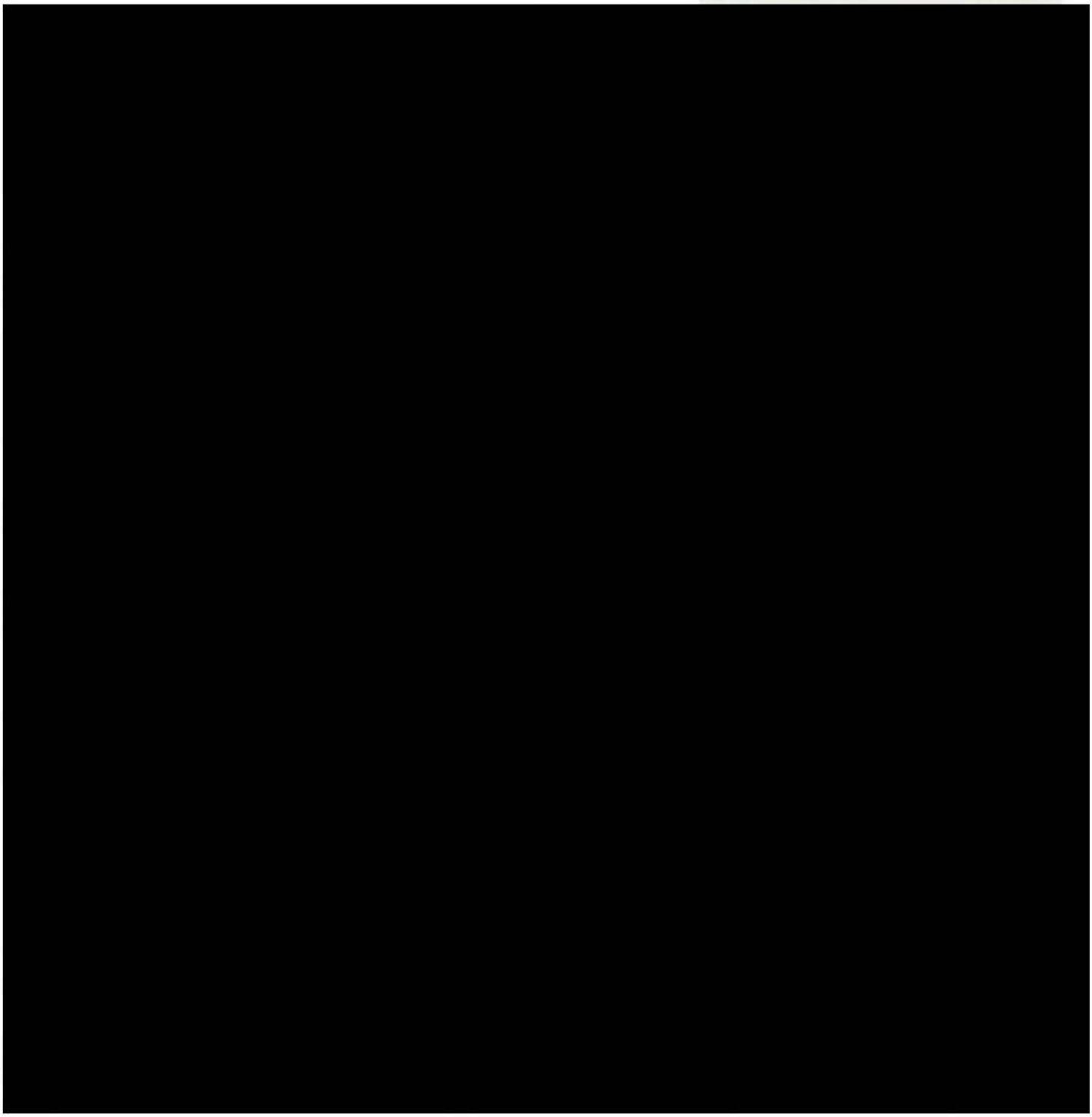

mean loss

This still left a concern that [redacted]




mean loss

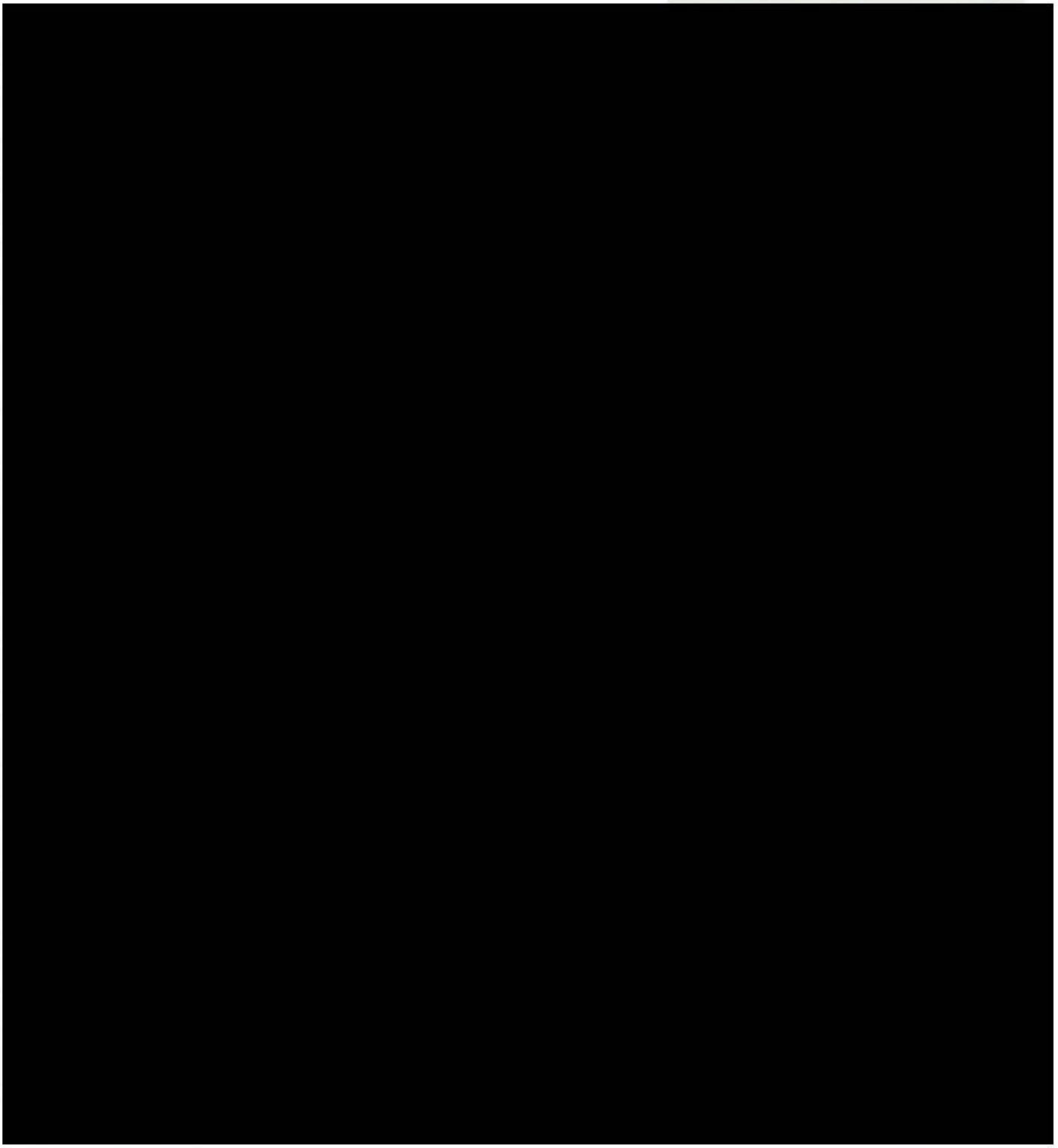
I looked at [redacted]
https://anthropic.slack.com/archives/[redacted]
Looks like [redacted]



How many unique things are memorized?

I made a plot of the number of unique tokens seen as parts of spans that are repeated > 50 times vs. n_layers. The y-axis here is normalized by n_params, since I feel like (n_unique_tokens_memorized / n_params) is an important number. It's definitely getting worse with scale

We see CMs of ~.5x just from duplicating .1% of the dataset 100 times (i.e. duplicating 1/50th as many tokens as there are params), and without exact-dedupe at scale there would be like 1/5th as many tokens memorized as there are params. Horrifying!
(Also, if you count with multiplicity, then like a quarter of the dataset is stuff that's past the memorization threshold.)



I'm using `DupesCounter` from memorization_stuff.py to make these graphs

Also, these ngrams follow a very tight zipf's law! So you can use something like this to extrapolate the effects of deduping at various frequency thresholds.


n_reps

(old, kind of hazy result)
I looked at a sample of cc.p8, and looked at how many unique strings show up >100 times as a function of dataset size. The exponent was 1.3 > 1. Soooo, that's pretty bad news bears for the giant models i.e. it suggests that a greater and greater fraction of capacity is getting devoted to memorization. It's also consistent with e.g. getting a 1.1x CM per model size increment in our dedupe scans.

What's getting memorized?

cc.p8 samples

The first cc.p8 stuff that gets removed is lists, lists, and lists. (Mostly website headers and footers and lots of semi-expanded links to other articles. Also boring copyright stuff, exhortations to share and subscribe or whatever etc. Also webspam that has somehow perfused the internet eh.)

These snippets all show up just 4 times in my shard of cc.p8, and still end up memorized, hm. If I didn't know these snippets were memorized, I'd say they look perfectly fine/innocuous.

```
wore many hats: a journalist, an activist, a publisher, a two-
<BREAK>
 than one month before parliamentary elections are to be held.
Politics, Middle East
<BREAK>
 a positive role. Wary of triggering a nationalist backlash, Erdogan
<BREAK>
.
"In Zimbabwe, he has been backing Mugabe throughout. That
<BREAK>
 Register of Probate positions once the two new positions are filled.
```

By way
<BREAK>
 emphasising that defending legitimate rights "does not mean slipping into war with
Israel".
<BREAK>
lfics In Shot Put And Discus; Slaby Brothers, Fischer Pace
<BREAK>

The pay of the Court Commissioner/Register in Probate was set by the
<BREAK>
 for whom, even though they have a good president [Mbeki] who
<BREAK>
 had fled across the Jordan where he found a refuge at the court of Baalis
<BREAK>
 400 (12th) and 800 relay teams.
The area also had the
<BREAK>
an fled.
Then followed a race across Europe in search of asylum, flying
<BREAK>
 That's the reason earlier this year the National Safety Council urged a total ban on
<BREAK>
 migrants rescued off Italian island
Incident comes a day after a boat laden
<BREAK>
. At one point, the population registered at 154 for a short time. A
<BREAK>
 - and without.
It is only his lack of a parliamentary seat that prevents
<BREAK>
 Counties, Brown County extended the offer to house Marinette County juveniles
<BREAK>
 nowhere to be seen.
It was a remarkable transformation from the dazed figure
<BREAK>
ney Heimerl pole vault 6th 8'0.
Kohls
<BREAK>
 achieved sustained economic growth during his tenure [EPA] Combined, these polarising
<BREAK>
 21'5.75. Creo VanBellinger of Suring was 8
<BREAK>
 are.
NSC said cell phone use while driving contributes to 6% of
<BREAK>
 blob out of the corner of his eye. When Tandy turned to face the
<BREAK>
 the dazed figure, bundled, handcuffed and blindfolded onto a jet
<BREAK>

Code samples

This is filtering for long memorized spans of code. Half of those are just licenses. Here are some random samples from the more interesting half.



Minified js lol

```
<BREAK>
</p>
                    <ul class="list-inline">
                        <li class="list-inline-item"><a class="text-color" href="#"><i class="ti-facebook"></i></a></li>
                        <li class="list-inline-item"><a class="text-color" href="#"><i class="ti-twitter-alt"></i></a></li>
                        <li class="list-inline-item"><a class="text-color" href="#"><i class="ti-google"></i></a></li>
                        <li class="list-inline-item"><a class="text-color" href="#"><i class="ti-linkedin"></i></a></li>
                    </ul>
                </div>
            </div>
        </div>
        <!-- teacher -->
        <div data-ref="mixitup-target" class="col-lg-4 col-sm-6 mb-5
<BREAK>
</h4>
                    </a>
                    <p>Teacher</p>
                    <ul class="list-inline">
                        <li class="list-inline-item"><a class="text-color" href="#"><i class="ti-facebook"></i></a></li>
                        <li class="list-inline-item"><a class="text-color" href="#"><i class="ti-twitter-alt"></i></a></li>
                        <li class="list-inline-item"><a class="text-color" href="#"><i class="ti-google"></i></a></li>
                        <li class="list-inline-item"><a class="text-color" href="#"><i class="ti-linkedin"></i></a></li>
                    </ul>
                </div>
            </div>
        </div>
<BREAK>
ok</h4>
                    </a>
                    <p>Teacher</p>
                    <ul class="list-inline">
                        <li class="list-inline-item"><a class="text-color" href="#"><i class="ti-facebook"></i></a></li>
                        <li class="list-inline-item"><a class="text-color" href="#"><i class="ti-twitter-alt"></i></a></li>
                        <li class="list-inline-item"><a class="text-color" href="#"><i class="ti-google"></i></a></li>
                        <li class="list-inline-item"><a class="text-color" href="#"><i class="ti-linkedin"></i></a></li>
                    </ul>
                </div>
            </div>
        </div>
        <!-- teacher -->
        <div data-ref="mixitup-target" class="col-lg-4 col-sm-6 mb-5
<BREAK>
h4>
                    </a>
                    <p>Teacher</p>
                    <ul class="list-inline">
                        <li class="list-inline-item"><a class="text-color" href="#"><i class="ti-facebook"></i></a></li>
                        <li class="list-inline-item"><a class="text-color" href="#"><i class="ti-twitter-alt"></i></a></li>
                        <li class="list-inline-item"><a class="text-color" href="#"><i class="ti-google"></i></a></li>
                        <li class="list-inline-item"><a class="text-color" href="#"><i class="ti-linkedin"></i></a></li>
                    </ul>
                </div>
            </div>
        </div>
        <!-- teacher -->
        <div data-ref="mixitup-target" class="col-lg-4 col-sm-6 mb-5
<BREAK>
```

Boring html from a heavily forked repo

```
<BREAK>
 Returns the current environment's envid.
static envid_t
sys_getenvid(void)
{
        return curenv->env_id;
}

// Destroy a given environment (possibly the currently running environment).
//
// Returns 0 on success, < 0 on error.  Errors are:
//      -E_BAD_ENV if environment envid doesn't currently exist,
//              or the caller doesn't have permission to change envid.
static int
sys_env_destroy(envid_t envid)
{
        int r;
        struct Env *e;
<BREAK>
 envid's env_status to status, which must be ENV_RUNNABLE
// or ENV_NOT_RUNNABLE.
//
// Returns 0 on success, < 0 on error.  Errors are:
//      -E_BAD_ENV if environment envid doesn't currently exist,
//              or the caller doesn't have permission to change envid.
//      -E_INVAL if status is not a valid status for an environment.

<BREAK>
// Set the page fault upcall for 'envid' by modifying the corresponding struct
// Env's 'env_pgfault_upcall' field.  When 'envid' causes a page fault, the
// kernel will push a fault record onto the exception stack, then branch to
// 'func'.
//
// Returns 0 on success, < 0 on error.  Errors are:
//      -E_BAD_ENV if environment envid doesn't currently exist,
//              or the caller doesn't have permission to change envid.
static int
sys_env_set_pgfault_upcall(envid_t envid, void *func)
{
<BREAK>
locate a page of memory and map it at 'va' with permission
// 'perm' in the address space of 'envid'.
// The page's contents are set to 0.
// If a page is already mapped at 'va', that page is unmapped as a
// side effect.
//
// perm -- PTE_U | PTE_P must be set, PTE_AVAIL | PTE_W may or may not be set,
//         but no other bits may be set.  See PTE_SYSCALL in inc/mmu.h.
//
// Return 0 on success, < 0 on error.  Errors are:
//      -E_BAD_ENV if environment envid doesn't currently exist,
//              or the caller doesn't have permission to change envid.
//      -E_INVAL if va >= UTOP, or va is not page-aligned.
//      -E_INVAL if perm is inappropriate (see above).
//      -E_NO_MEM if there's no memory to allocate the new page,
//              or to allocate any necessary page tables.

<BREAK>
```

idk

```
<BREAK>
, 0, 0.1)',
                lineWidth: 1,
                drawBorder: true,
                drawOnChartArea: true,
                drawTicks: true,
                tickMarkLength: 10,
                zeroLineWidth: 1,
                zeroLineColor: 'rgba(0,0,0,0.25)',
                zeroLineBorderDash: [],
                zeroLineBorderDashOffset: 0.0,
                offsetGridLines: false,
                borderDash: [],
                borderDashOffset: 0.0
        },

        // scale label
        scaleLabel: {
                //
<BREAK>
.global;
                var size = valueOrDefault(options.fontSize, globalDefaults.defaultFontSize);
                var style = valueOrDefault(options.fontStyle, globalDefaults.defaultFontStyle);
                var family = valueOrDefault(options.fontFamily, globalDefaults.defaultFontFamily);

                return {
                        size: size,
                        style: style,
                        family: family,
                        font: helpers.fontString(size, style, family)
                };
        }

        function parseLineHeight(options) {
                return helpers.options.toLineHeight(
                        helpers.valueOrDefault(options.lineHeight, 1.2),
                        helpers.valueOrDefault(options.fontSize, defaults.global.defaultFontSize));
        }


<BREAK>
 * Get the padding needed for the scale
                 * @method getPadding
                 * @private
                 * @returns {Padding} the necessary padding
                 */
                getPadding: function() {
                        var me = this;
                        return {
                                left: me.paddingLeft || 0,
                                top: me.paddingTop || 0,
                                right: me.paddingRight || 0,
                                bottom: me.paddingBottom || 0
                        };
                },

                /**
```

This file is super long

```
<BREAK>
 Red = 0.0;
            Green = (Wavelength - 440) / (490 - 440);
            Blue = 1.0;
        }else if((Wavelength >= 490) && (Wavelength<510)){
            Red = 0.0;
            Green = 1.0;
            Blue = -(Wavelength - 510) / (510 - 490);
        }else if((Wavelength >= 510) && (Wavelength<580)){
            Red = (Wavelength - 510) / (580 - 510);
            Green = 1.0;
            Blue = 0.0;
        }else if((Wavelength >= 580) && (Wavelength<645)){
            Red = 1.0;
            Green = -(Wavelength - 645) / (645 - 580);
            Blue = 0.0;
        }else if((Wavelength >= 645) && (Wavelength
<BREAK>
 factor = 0.3 + 0.7*(Wavelength - 380) / (420 - 380);
        }else if((Wavelength >= 420) && (Wavelength<701)){
            factor = 1.0;
        }else if((Wavelength >= 701) && (Wavelength<781)){
            factor = 0.3 + 0.7*(780 - Wavelength) / (780 - 700);
        }else{
            factor = 0.0;
        };
<BREAK>
ChunksPlugin is enabled by default and replaced
 * deprecated CommonsChunkPlugin. It automatically identifies modules which
 * should be splitted of chunk by heuristics using modulo duplication count and
 * module category (i. e. node_modules). And splits the chunks...
 *
 * It is safe to remove "splitChunks" from the generated configuration
 * and was added as an educational example.
 *
 * https://webpack.js.org/plugins/split-chunks-plugin/
 *
 */
```



Literally lorem ipsum lol

How harmful is memorization?

We're consistently seeing insane CMs like 1.5x from just removing 10% of the data, which means that 10% is doing a lot more harm than if it was just ignorable filler.

Commented [3]: I think it's like 1.4x for 2% on code :monkey: [Author:kipply c].[Creator:1297307460].[Date:8/16/2023 12:24:00 AM]

Danny's paper

Danny published a paper looking into memorization last year that got a similar result (he was seeing as much as a 2x model size hit from just training on 10% highly repeated data!)

https://arxiv.org/abs/2205.10487

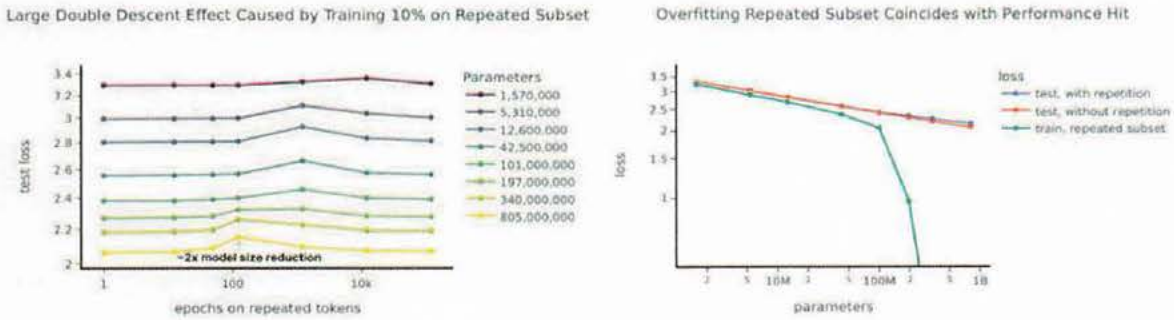


Figure 2 Models of different sizes show a degradation in performance at a specific range of repeats that shrinks with model size (left panel). At its peak the degradation sometimes reaches the equivalent of a 2x decrease in model size. The right panel shows that divergence (blue line) from a healthy, straight scaling law (red) lines up with when the models start to dramatically overfit the repeated subset (green curve). The blue line on the right corresponds to a vertical slice of models in the left diagram trained on the repeated subset for 120 epochs. All these models were trained on 90% unique data and 10% repeated tokens.

Danny's paper gives a very reasonable argument for why this might happen:

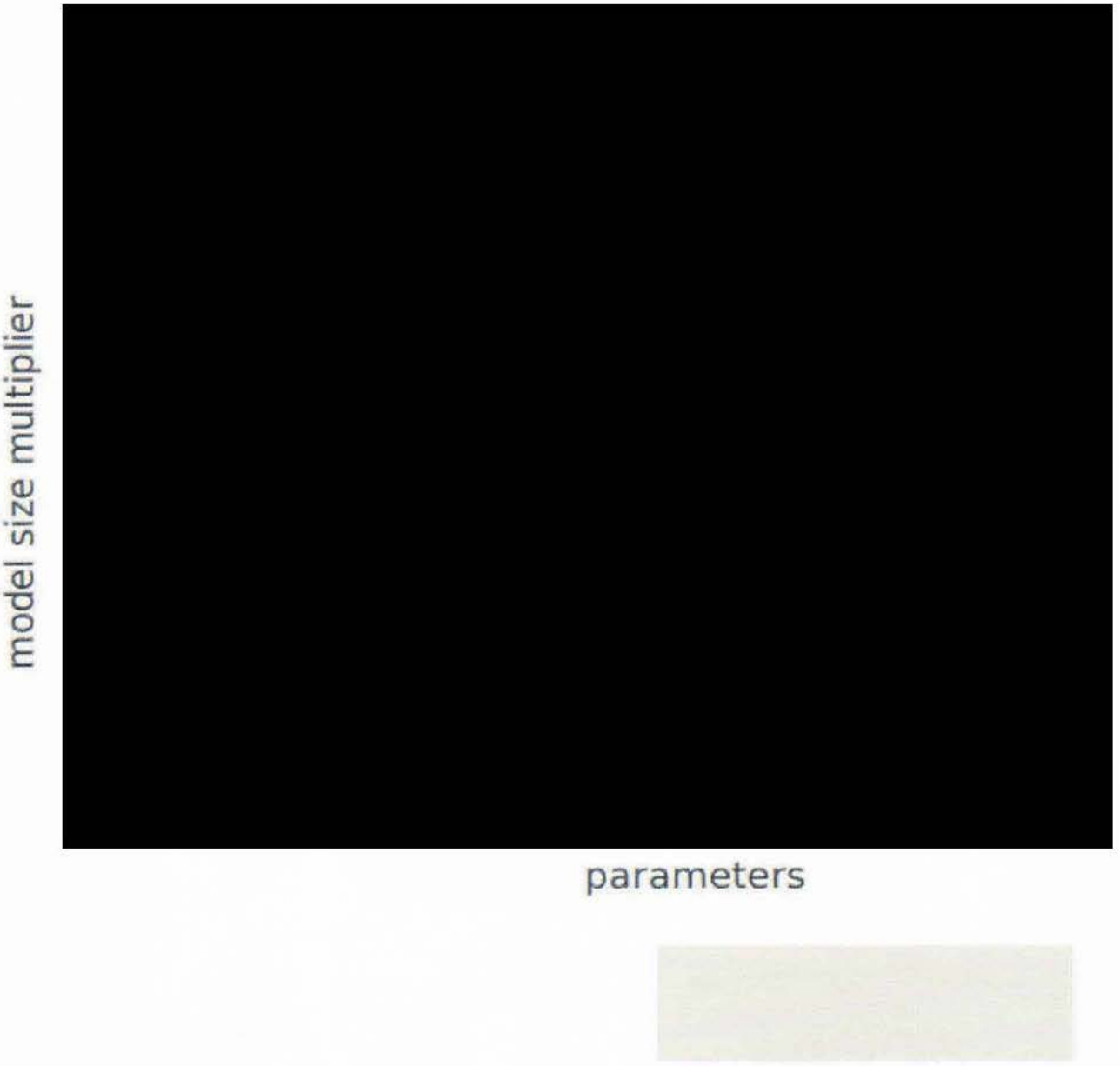
5.1 Why does repeating a small fraction of data damage performance so much?

We showed that a dataset with only 10% repeated tokens can reduce model performance by an effective 2x in parameter count, much more than if that 10% of the data had simply never been trained on. The repeated data thus degrades model performance out of proportion to its share in the dataset. Why does this occur, and why only for a specific amount of repetition? One plausible hypothesis comes from looking at the model's "incentives" to memorize vs generalize. To informally explore this hypothesis consider the following rough numbers, a 800M parameter model typically has a loss of roughly 2.0 nats/token, a 400M parameter model has a loss of roughly 2.2 nats/token, and fully memorized data will have a loss of 0 nats/token. Now suppose a 800M model is trained on 90% unique data and 10% tokens consisting of repeated data. We can ask whether it is a "good tradeoff" for the model to memorize the repeated data (leading to 0 loss on 10% of the dataset), at the cost of degrading performance by the equivalent of a 2x multiple in model size (which raises loss on the other 90% from 2 to 2.2). Some simple arithmetic suggests that it is: $0.9 * 2.2 + 0.1 * 0 = 1.98 < 2.0$. Another way to say this is that zero loss is such a huge drop compared to the differences in entropy between model sizes that driving the loss to zero on even a tiny subset can incentivize enormous degradation in quality.

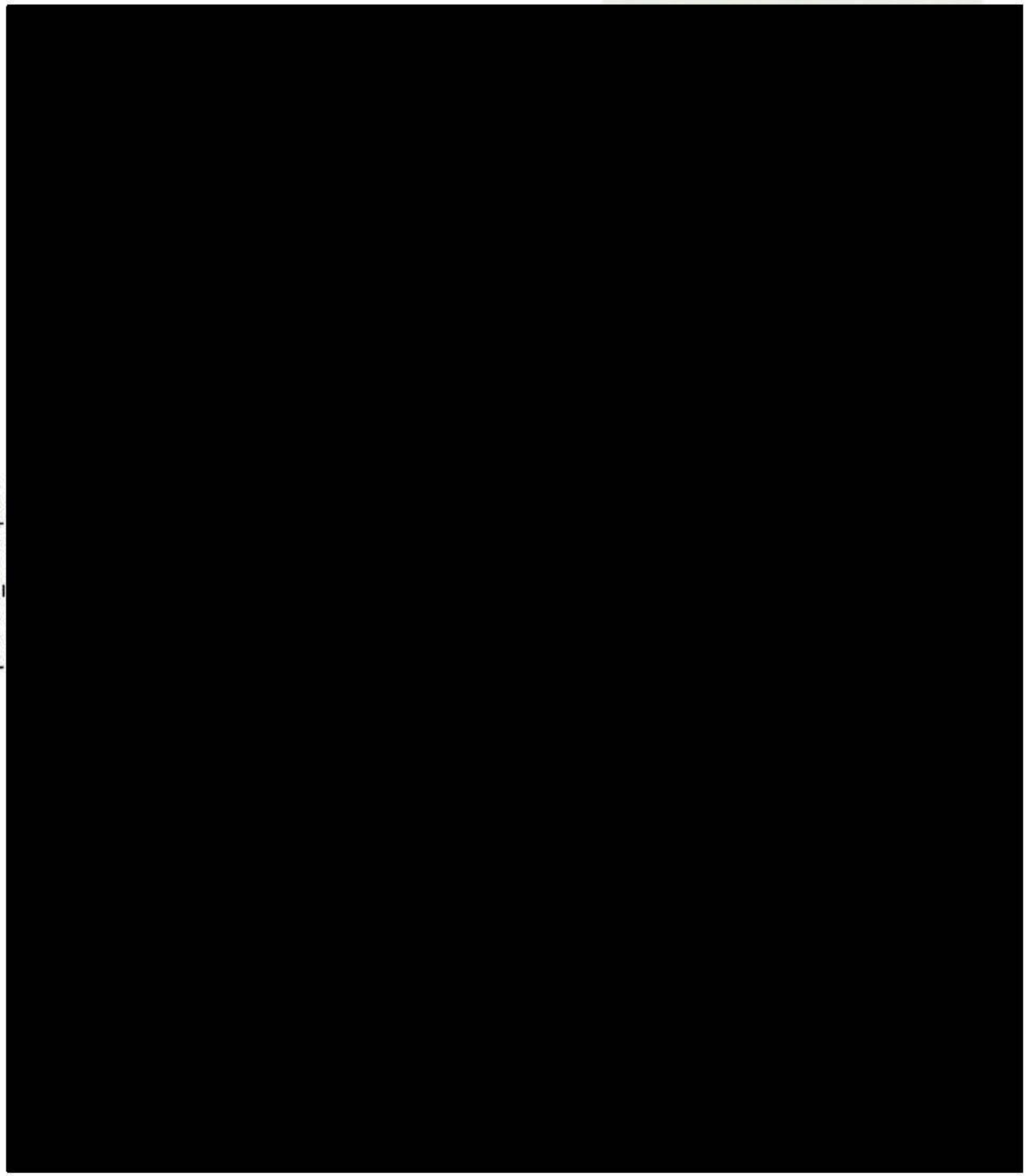
This however leaves open the question of when this tradeoff is necessary or possible -- and here is where the double descent phenomenon comes in. If a lot of data is repeated only a few times (say 5% of the data repeated 2x) then the model may not have the capacity to memorize it, and also does not see it enough times during training to do so. If a tiny amount of data is repeated very many times (say 0.01% of the data repeated 1000x), then the model will memorize it, but because it is so small the model need not use much capacity to do so, so the degradation in quality will likely be small. There is a range in the middle where the data can be memorized *and* doing so consumes a large fraction of the model's capacity, and this may be where the peak of degradation occurs.

From the perspective of the training loss, it's a very worthwhile trade to devote a ton of capacity to memorization!

I tried fitting to Danny's results based on his hypothesis/model. They were pretty okay for the bigger models (biggest is [redacted] but pretty terrible for the smallest models, eh


model size multiplier
parameters




param_multiplier

I also tried fitting a model like this to [redacted], and it was pretty bad (it suggested [redacted] than it was)

More Synthetic Dupes Experiments

Ethan and I have done a bunch of follow-ups replicating Danny's experiment (most notably we switched to chinchilla scaling).

First of all we were able to replicate some terrible CMs (along with Danny's double descent). [redacted]

[redacted]

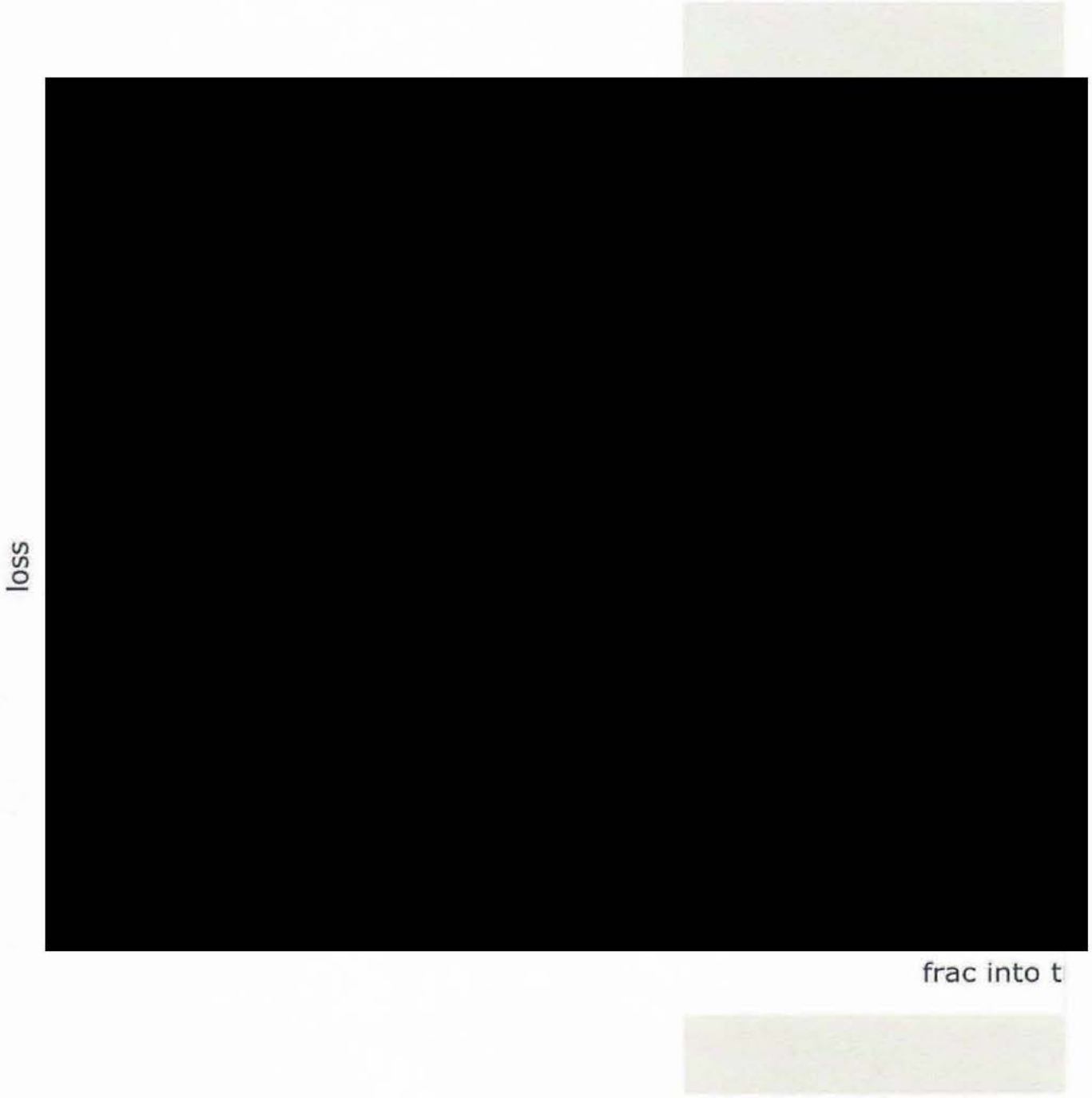



```
18:45:24,396 spectra.analysis.fitting:149 2194377 [00000] Extrapolating CM; extend your ba
18:45:24,399 spectra.analysis.fitting:149 2194377 [00000] Extrapolating CM; extend your ba
18:45:24,401 spectra.analysis.fitting:149 2194377 [00000] Extrapolating CM; extend your ba
18:45:24,403 spectra.analysis.fitting:149 2194377 [00000] Extrapolating CM; extend your ba
```



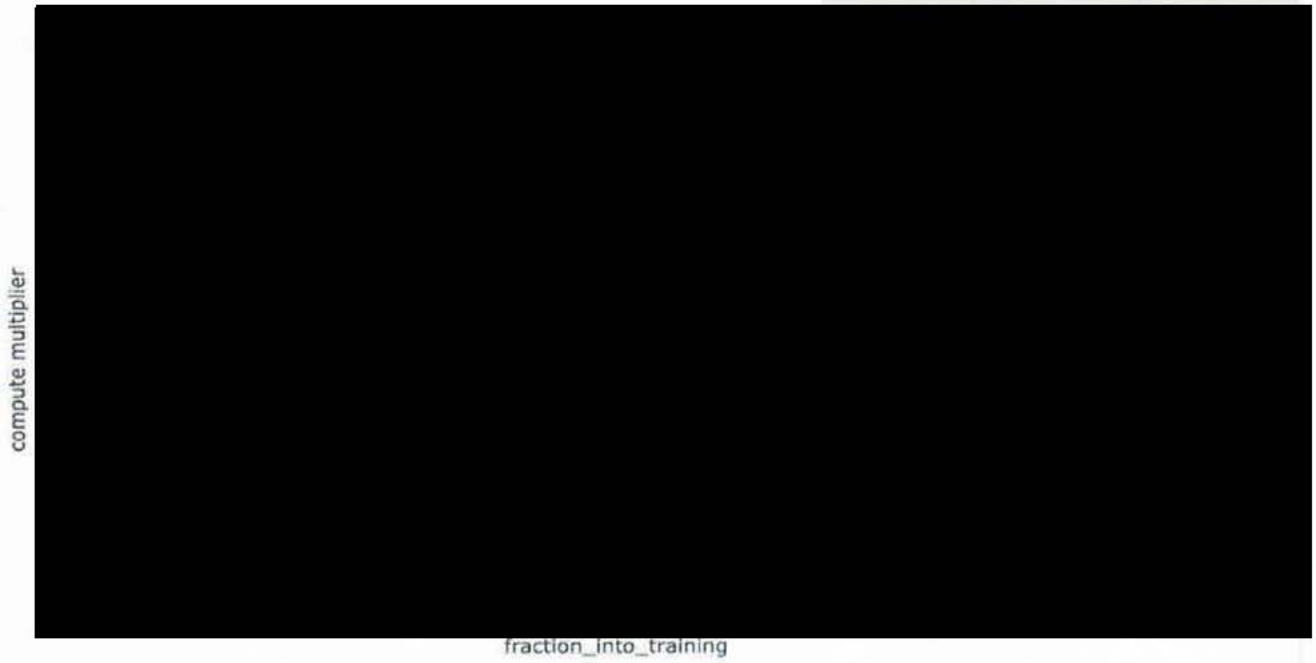
Second, it looks like the losses on the sections that are [redacted]

https://anthropic.slack.com/archives/[redacted]


loss
frac into t

Note how e.g. [redacted]

[redacted]
https://anthropic.slack.com [redacted]


compute multiplier
fraction_into_training


compute multiplier

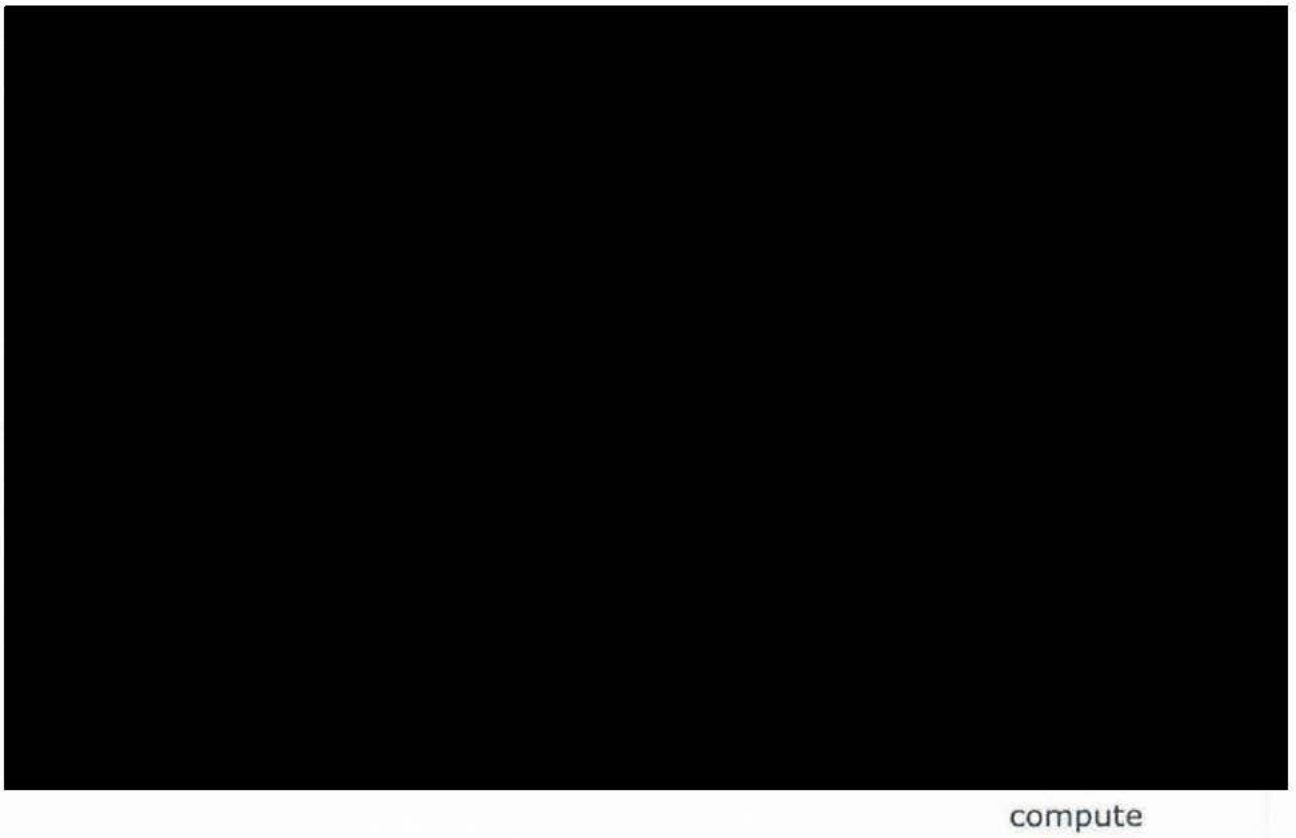
[redacted]



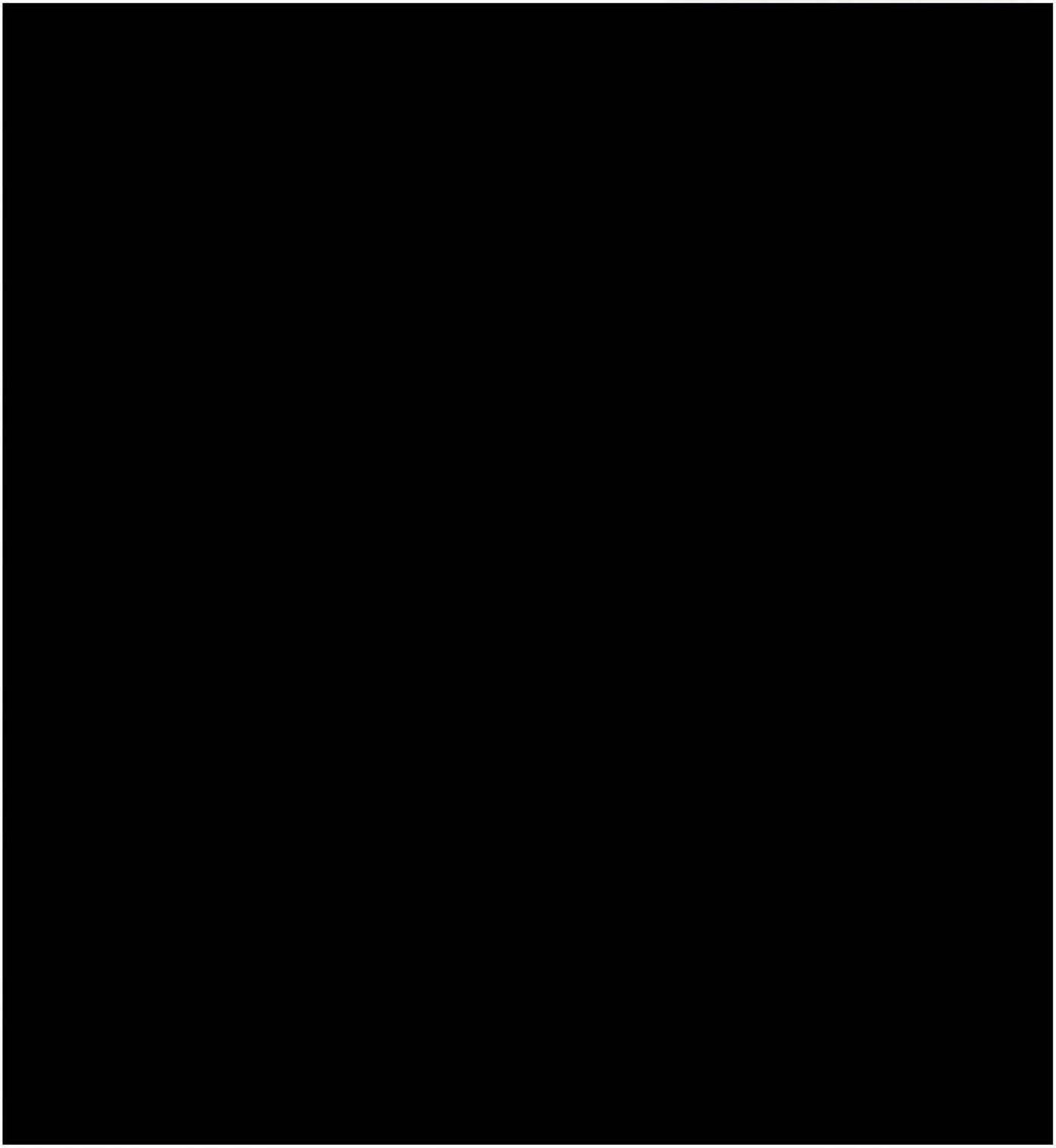
So this is *not* a hard tradeoff to make from the point of the train loss at all, it's just that optimization is hard

One thing that really surprised me is that, [redacted]

https://anthropic.slack.com/archives/[redacted]

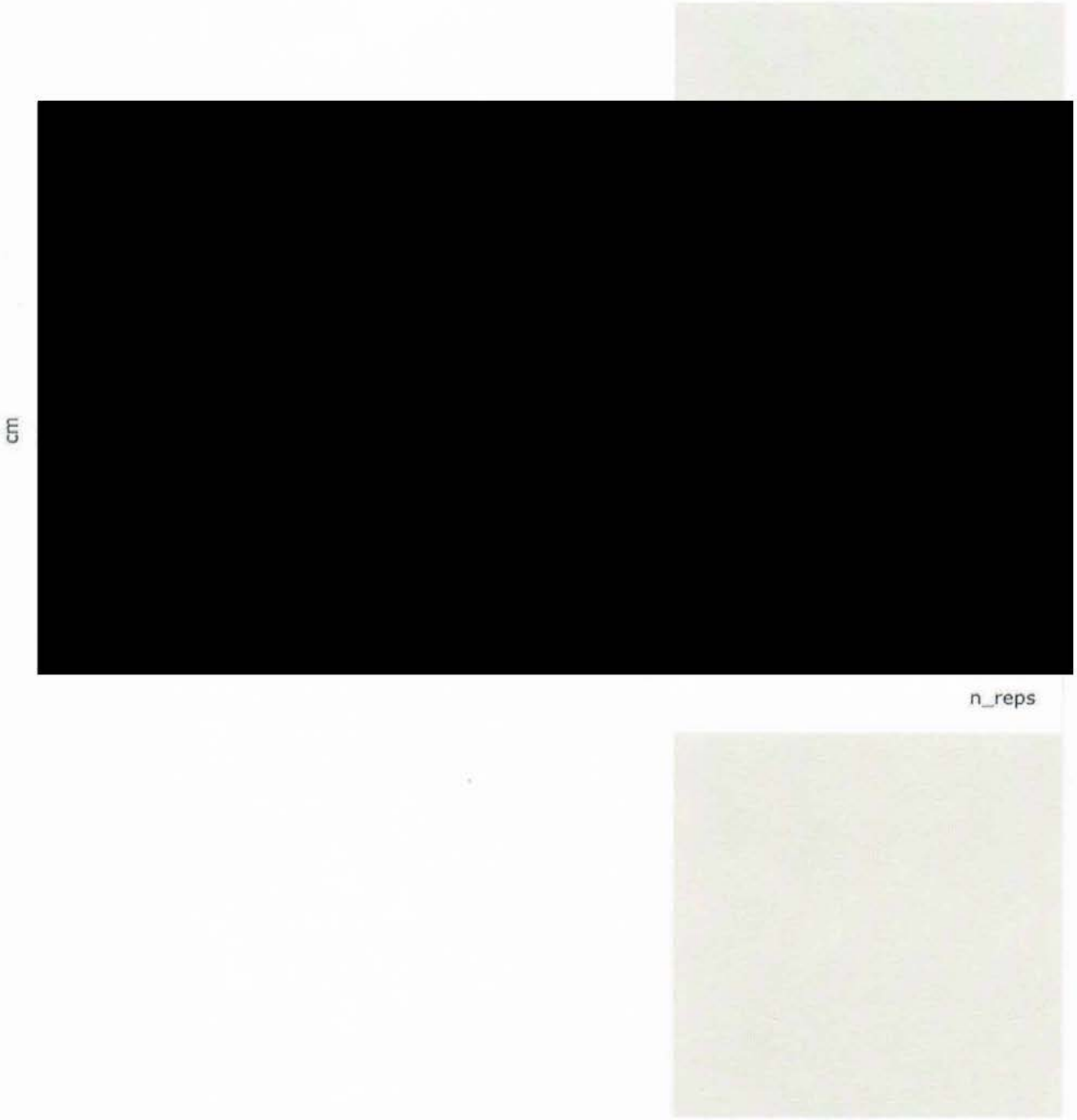


Indeed, it looks like [redacted]

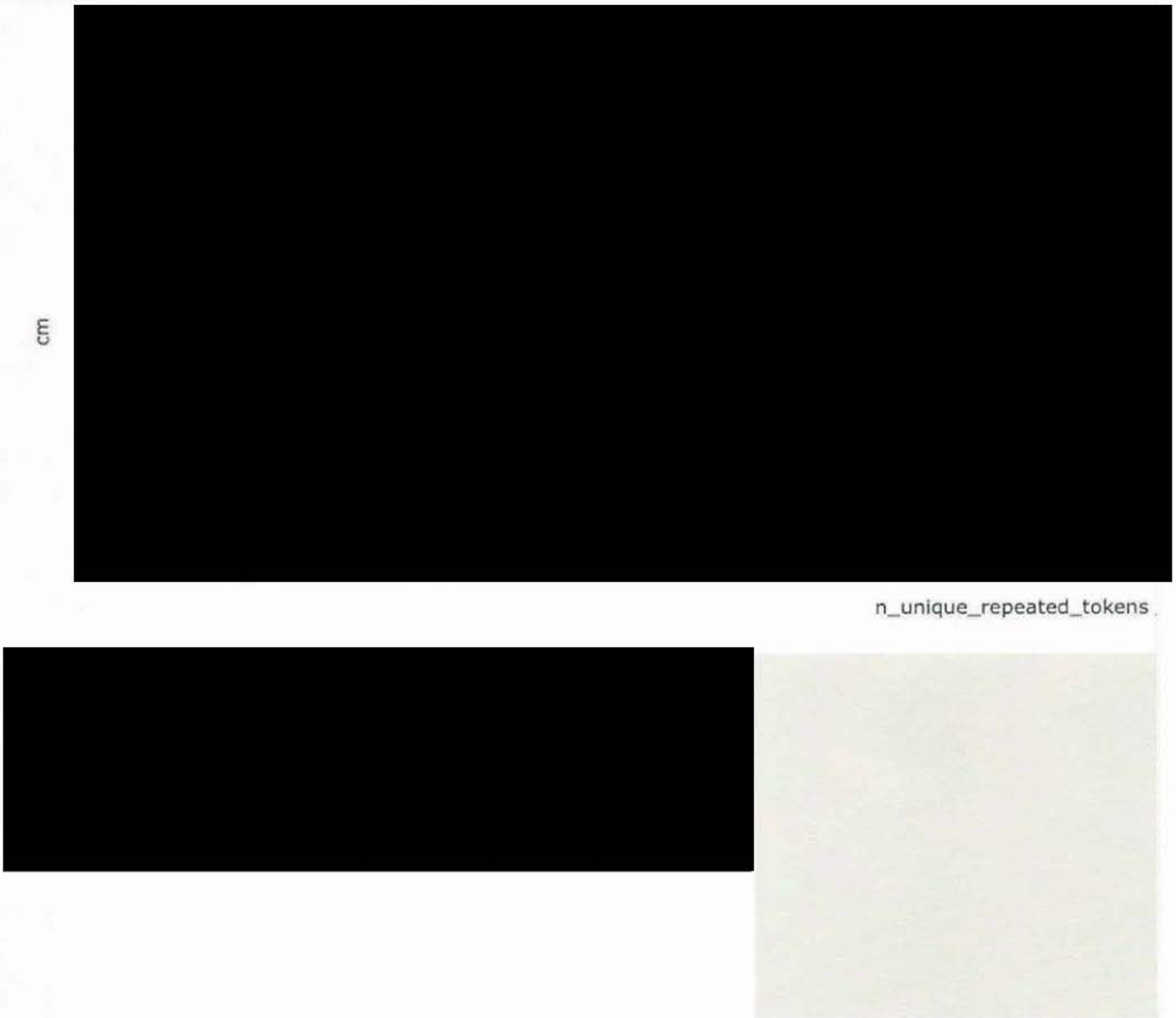


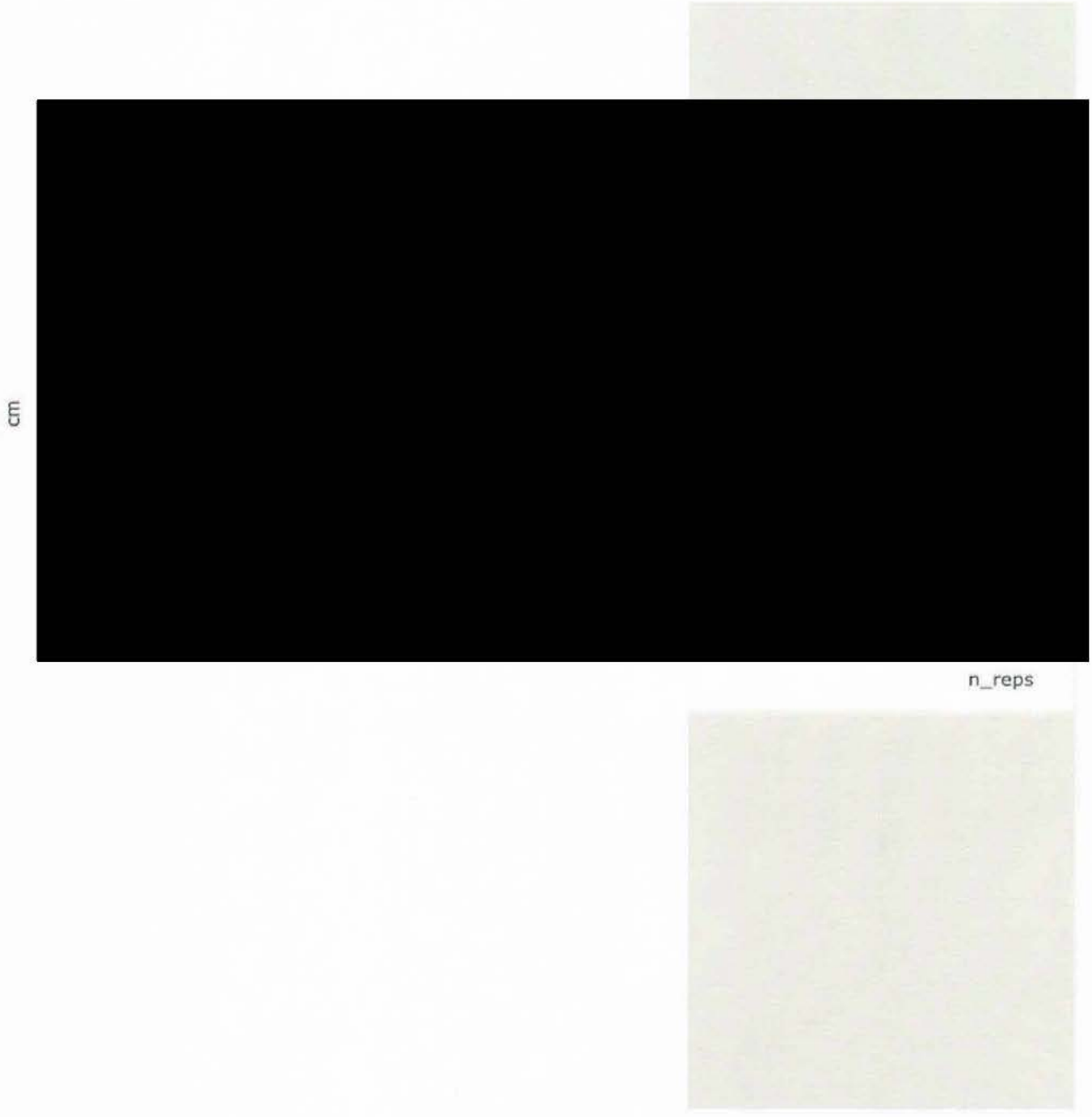
I rerar [redacted]

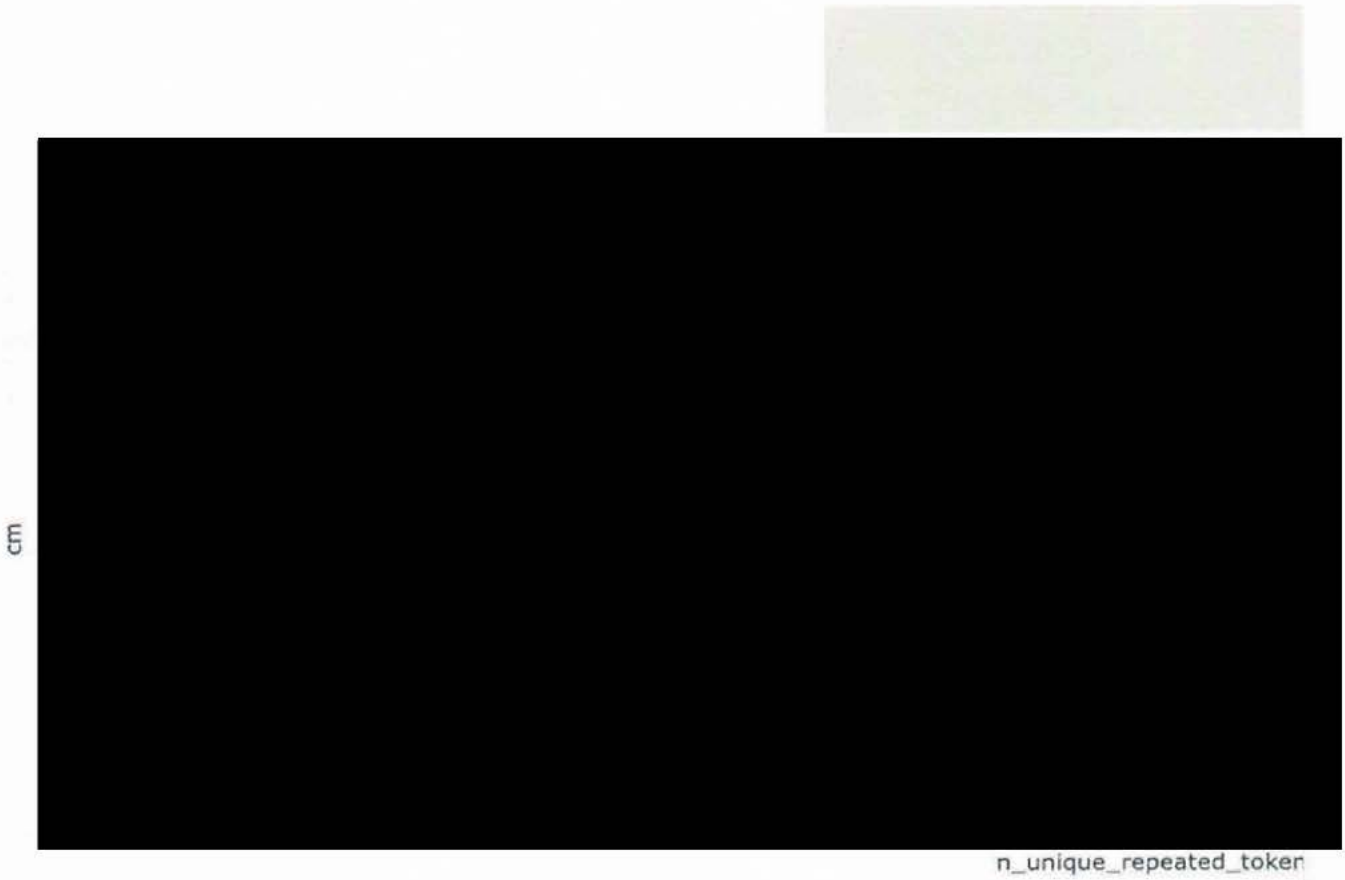



cm
n_reps

Reparameterized:




cm
n_reps

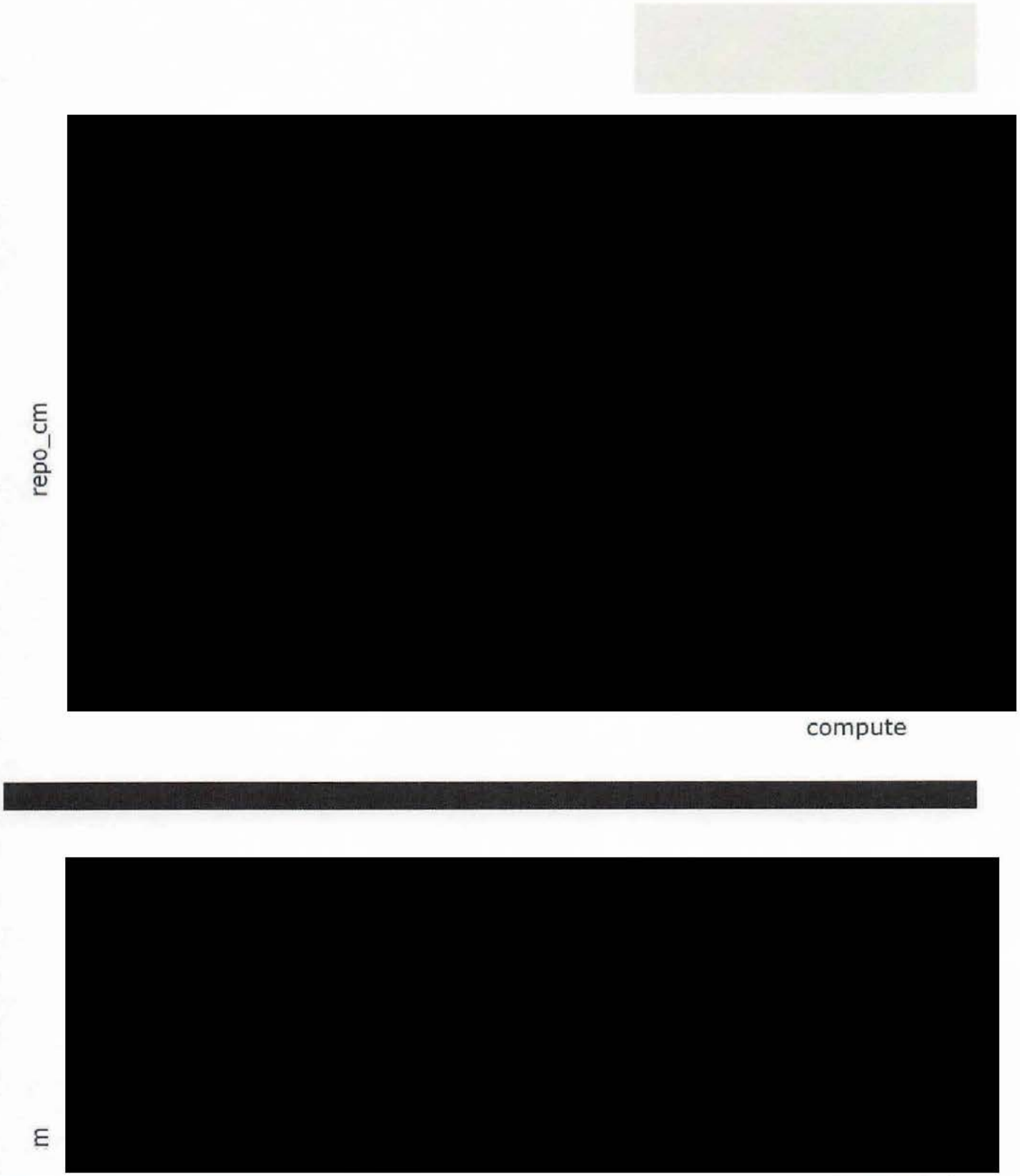


Forgetting

Sam suggested [redacted]

Kipply did some recal experiments, training on [redacted]

https://anthropic.slack.com/archives/[redacted]
I don't think we know if [redacted]


repo_cm
compute
m

Danny and Nick looked [redacted]

[redacted]

From a thread in # pretraining-org-updates | Jun 20th, 2022 | View reply

2 1

(sadly, if you try to follow that link, it's been retentioned T.T)

Dropout

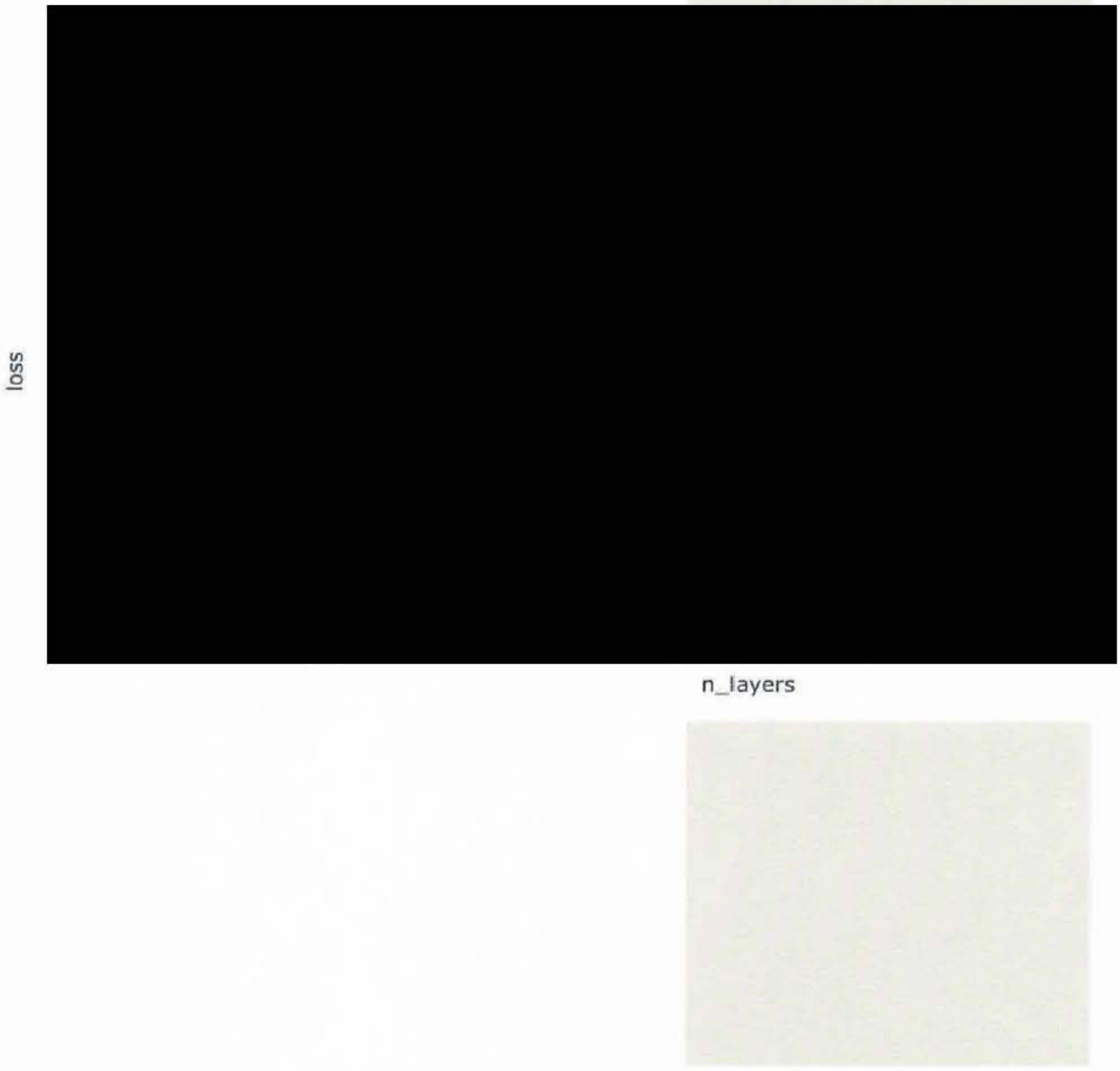
Mlucy is the most recent person to experiment with dropout. [redacted]

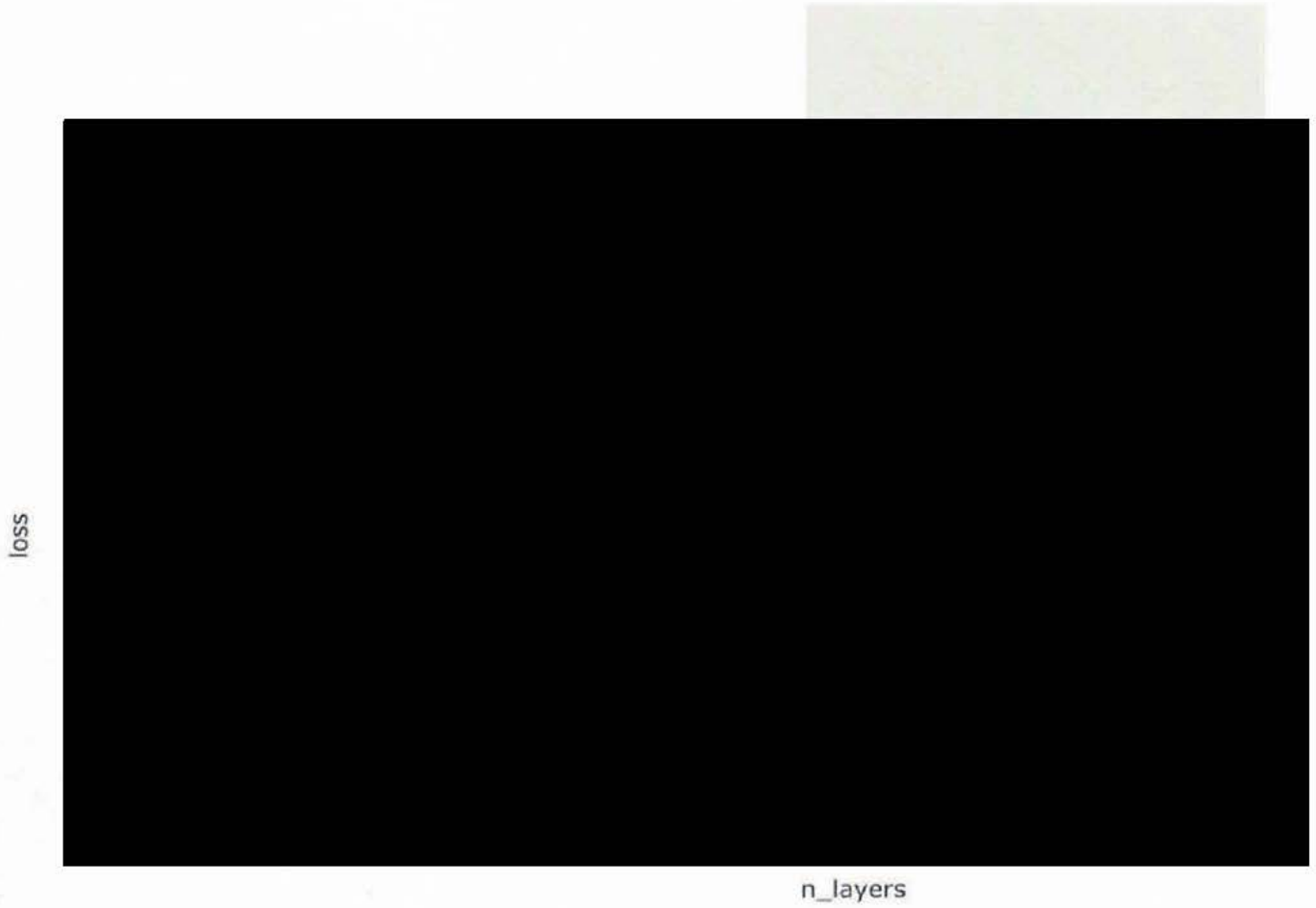
Estimating the CM hit for tern/teal/[redacted]

[redacted]

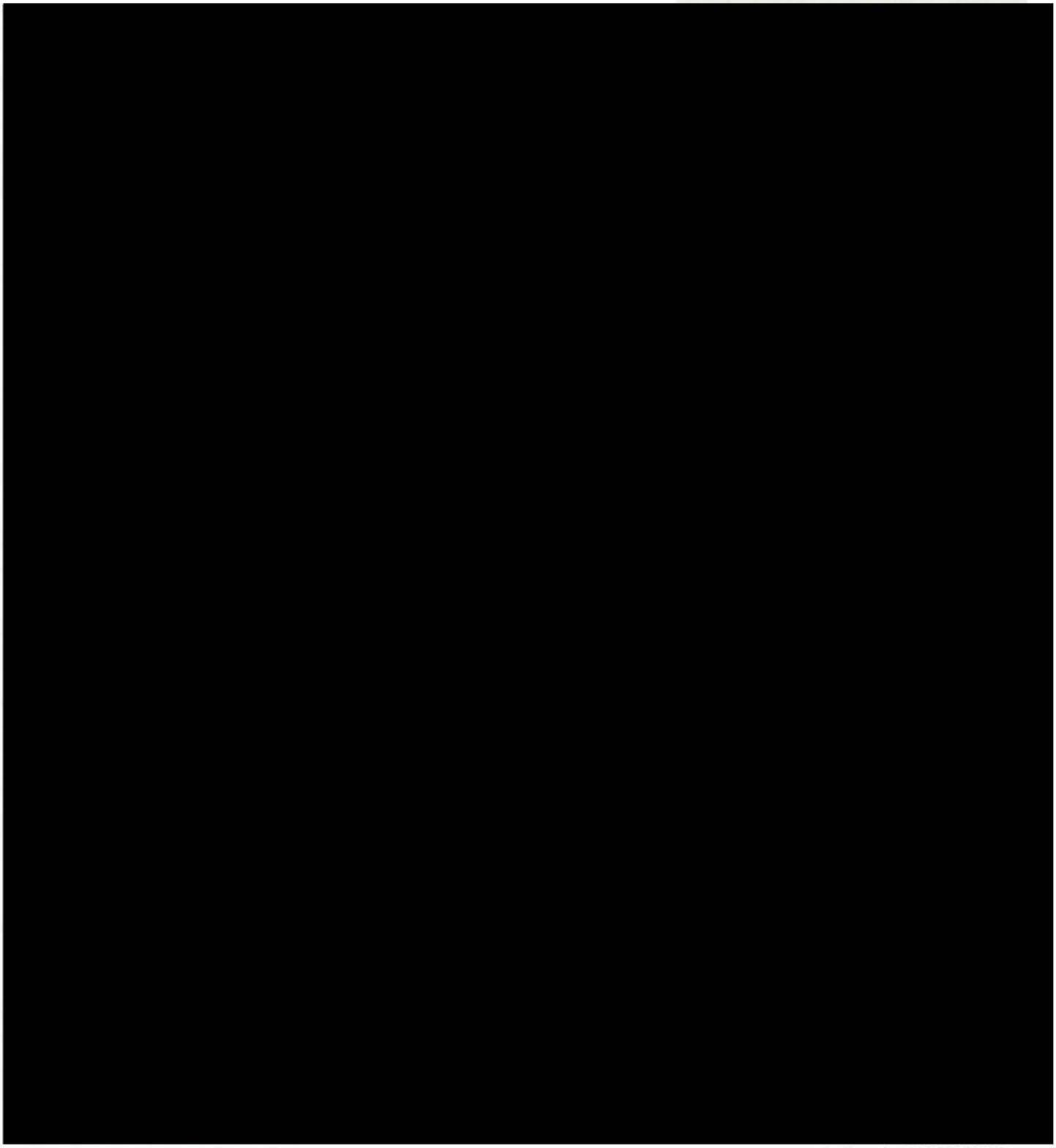
Memorization is [redacted]

[redacted]


loss
n_layers



One extreme example is [redacted]



Memorization is [redacted]

[redacted]



How much capacity does it take to memorize

[redacted]
(graaaaaphs)

A newer paper suggests the more relevant number is the number of sequences to memorize, rather than the number of tokens 😟. Anyway, clearly we should just do our own scans and plot the results.
Edit: I think I was referring to this paper https://openreview.net/forum?id=8JCg5xJCTPR, which, yeah, has a sqrt, but it's sqrt(nN) T.T, so linear lol

A quibble about "capacity" – Robert objects to me talking about [redacted]

Mechanistic explanation of memorization

How exactly does memorizing "use up capacity"? Idk. I asked Danny, Henighan, and Josh, and they didn't know how. Josh gave some evidence that [redacted]

Commented [4]: were these random tokens, or real data? [Author:Robert Lasenby],[Creator:1297307460],[Date:8/16/2023 12:59:00 AM]

Commented [5]: real [Author:Dawn Drain],[Creator:1297307460],[Date:8/16/2023 1:11:00 AM]

Todo's

Oldish notion todos: https://www.notion.so/anthropic/memorization-todo-405a97e67efb4d78aeeddb18ff7e1720

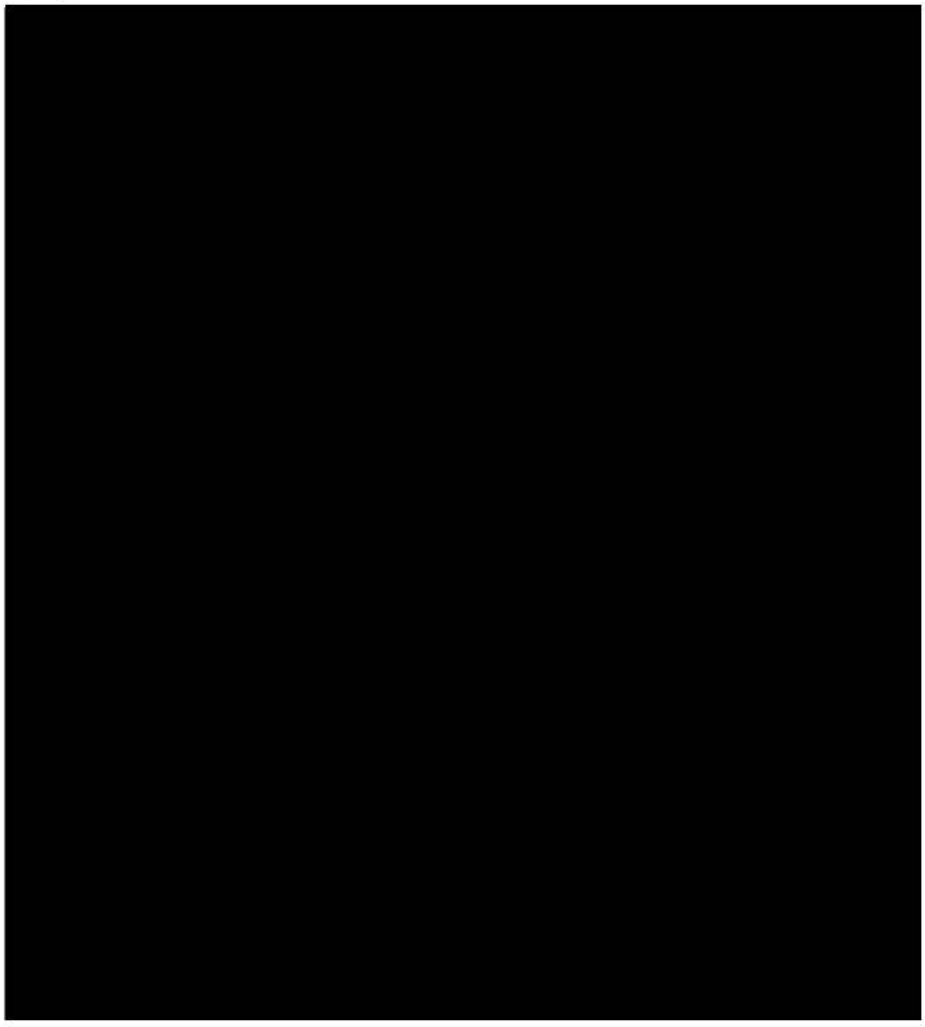
Teal prep todos:
https://docs.google.com/document/d/1ImZyi02h02LB3wYGmokyZNsu_KFS0OeNYtZWPN4-Rjw/edit

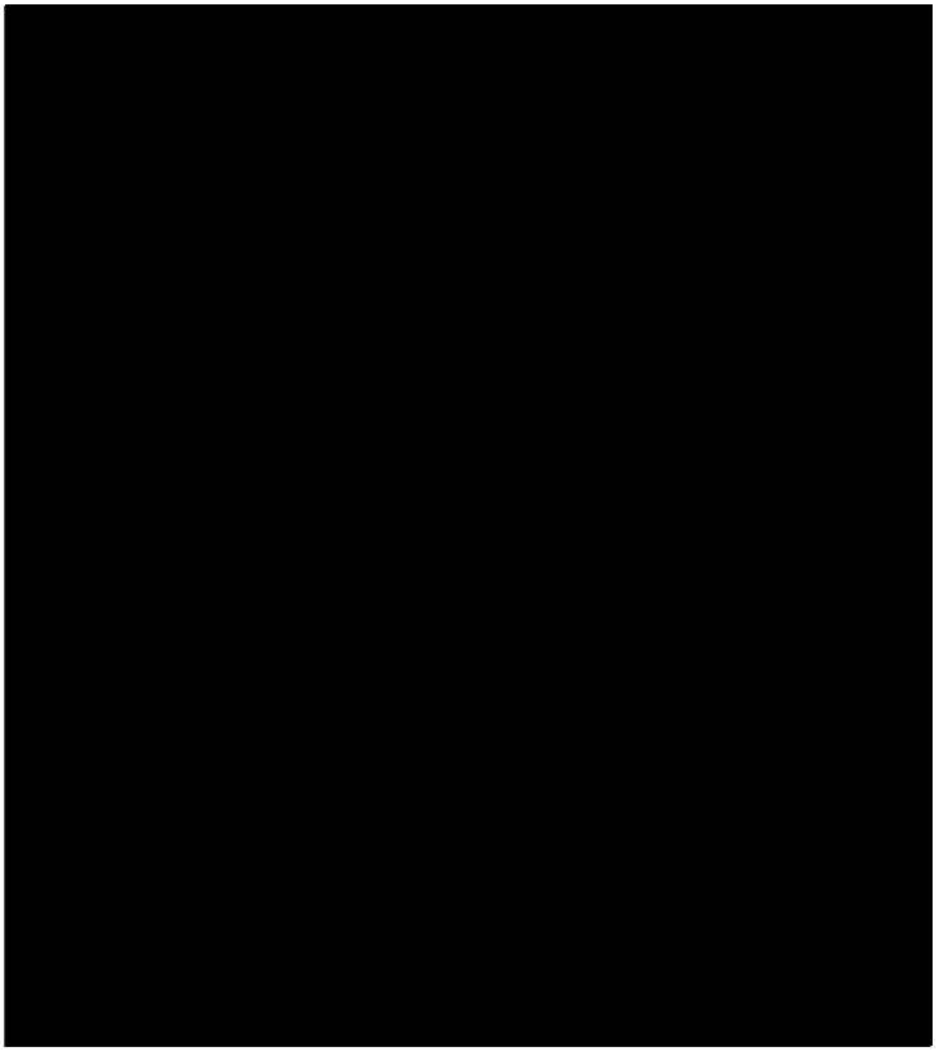
(Gonna rewrite my todo's here out of paranoia about notion, sigh)

Oct 13, 2023

1. Shingled fuzzy file-level dedupe
    a. This is especially impactful for code (especially the js and css datasets that Gyula crawled), and we might want to go back and run it on all our code data
2. "Paragraph" level dedupe
    a. They do this at google! We should do it
3. Memorization hook
    a. Throughout training monitor to see if large swathes of the dataset are being memorized, and later iterate on strategies for removing them
4. Take a random sample of 1000 files/sequences, look at what's memorized, see how much overlaps with the stuff we're deduping, and iterate on fixing it
5. Just run standard fuzzy dedupe on cc.documents*, pile of law etc.
    a. This is like 3% of the data mix where we just forgot to run dedupe
6. Maybe go back to excising (just don't excise quotes)
7. Better vis for exact dedupe
8. Just come up with better heuristics for excising garbage from common crawl

Sep 13, 2023





Current state as of Sep 20

I think we'll probably want [redacted] Hopefully a candidate aggressively-deduped dataset will be ready today T.T, and I can share better statistics

(dawn)
[redacted] but I think I have a workaround

[redacted]

(pranay)
[redacted]

(kipply)
[redacted]

(ethan and ansh)
[redacted]

(jamie)

Commented [6]: I don't think this is the highest priority, but I'm [redacted] @anthropic.com [redacted] @anthropic.com who might want to try this. I think it's super quick to try, just tweaking the LR schedule a bit.
1 total reaction
Ansh Radhakrishnan reacted with 👍 at 2023-09-21 11:09 AM [Author:Nicholas Joseph],[Creator:1297307460],[Date:9/21/2023 5:16:00 PM]

[redacted]