AI Model Management

Michael McCool, Geoff Gustafson,
Sudeep Divakaran, Muthaiah Venkatachalam

Intel

25 September 2024, TPAC 2024 Breakout

Agenda

- Problem Statement (10m)
- Issues and Constraints (15m)
  - Discussion
  - Prioritization
- Discussion of Alternatives (20m)
  - Possible Solutions
  - Pros and Cons
- Next Steps (5m)

Problem Statement

AI models:

- Can be very large
- Will often be shared
- Are often updated

Current Same-Origin Storage Partitioning policy:

- Preserves privacy
- ***Works for images:*** large, but usually not shared
- ***Works for software libraries:*** often shared, but usually small

Use Cases for Large Models

- Language Translation
    - Content may be private/confidential
    - Offline usage during travel
- Meeting captions
    - Generally private/confidential
    - Assuming peer-to-peer connection
- Background Removal
    - Source images private
- Video creation and editing
    - Bandwidth limited
- Written language recognition
    - Offline usage during travel
- Personal assistant
    - Private/confidential
    - Larger models needed to support reasoning and planning

Why Run AI on the Client?

Pros:

- Latency
- Offline

Cons:

- Model size limitations
- Download time
- Storage costs

Either way, depending:

- Performance
  - Server vs. client hardware
  - Network latency & bandwidth
- Efficiency
  - Server power & network
  - Battery power
- Privacy
  - Work on local data: pro!
  - Fingerprinting, tracking: con!

Model Size vs. Download Time

Average Home Network Speeds

- 45 Mbps – Baseline
- 90 Mbps – Global (optimistic)
- 216 Mbps – US

Sources:

- Speedtest.net
- USA Today: What is a Good Internet Speed

Maximum download in 1 minute:

- Baseline: 337.5 MB
- Global: 675 MB
- US: 1620 MB

Time to download Phi-3-mini:

- 3.8B bfloat16 parameters
- 2*3.8B = 7.6 GB
- Baseline: 22.5 minutes
- Global: 11.3 minutes
- US: 4.7 minutes

Existing APIs and Experiments

Same-Origin:

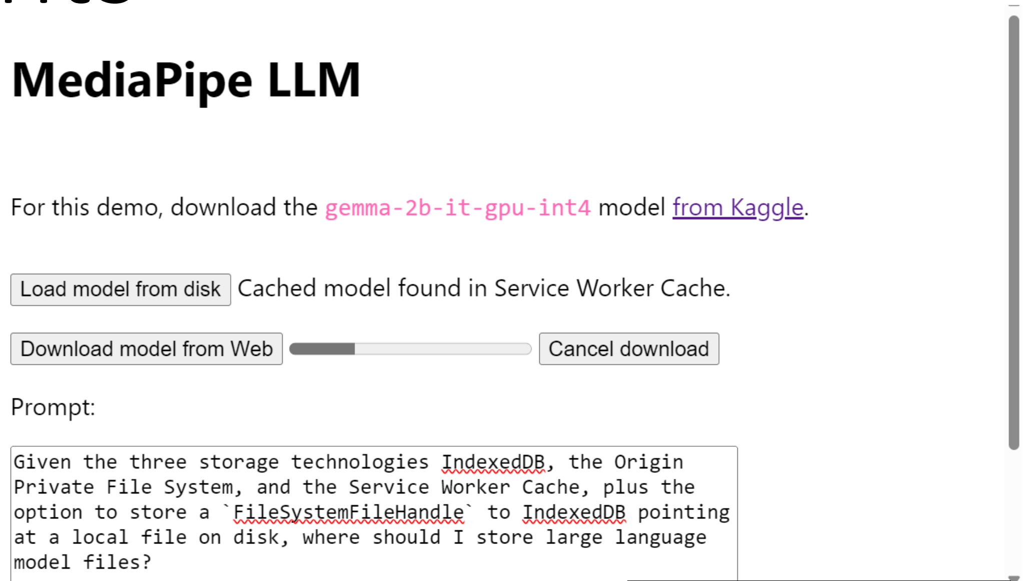
- HTTP Cache (browser and CDN)
- Cache API
- IndexedDB API
- Origin Private File System API

Cross-Origin:

- File System Access API
  - Requires user approval on first use using file selector UI
  - Get handle for reloads, persist in IndexedDB

Google Demo: [Cache AI models in the browser]()

See also: [Background Fetch]()



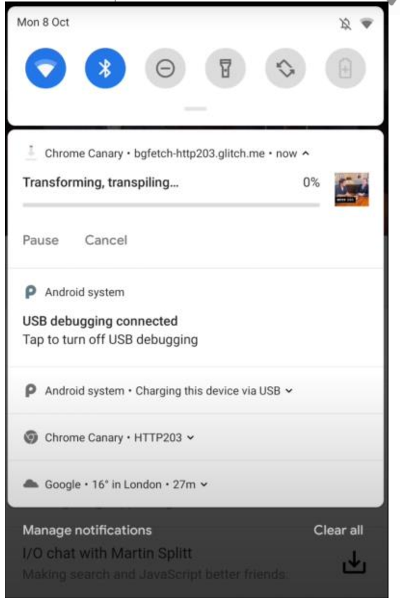
MediaPipe LLM

For this demo, download the `gemma-2b-it-gpu-int4` model [from Kaggle]().

Cached model found in Service Worker Cache.

Prompt:

Given the three storage technologies IndexedDB, the Origin Private File System, and the Service Worker Cache, plus the option to store a `FileSystemFileHandle` to IndexedDB pointing at a local file on disk, where should I store large language model files?



Mon 8 Oct

Chrome Canary • bgfetch-http203.glitch.me • now

Transforming, transpiling... 0%

Android system

USB debugging connected
Tap to turn off USB debugging

Android system • Charging this device via USB

Chrome Canary • HTTP203

Google • 16° in London • 27m

I/O chat with Martin Splitt
Making search and JavaScript better friends.

Caching Desired Properties

1. **Reduce Latency:** Fetch model from cache upon second use
2. **Reduce Bandwidth:** Avoid unnecessary downloads
3. **Reduce Storage:** Reuse space as much as possible
4. **Preserve Privacy:** Avoid tracking

Security and Privacy Considerations

- ***Current browsers generally implement only per-origin local caches:***
  - General “Storage Partitioning” policy
- Cross-site privacy risk based on cache timing analysis:
  - Site A can figure out if a user visited Site B
- Per-origin caches tolerable for “typical” (non-AI) web resources:
  - Sharing rate for images is low in practice
  - Files that are often shared tend to be small, e.g. script libraries
  - **BUT AI Models are large *and* potentially shared**

Issue Starter Pack

AI Model Management · Issue #15 · w3c/tpac2024-breakouts

- Background model download and compilation.
- Model naming and versioning
- Allowing for model substitution when useful
- Common interface for downloadable and “platform” models
- Storage deduplication
- Model representation independence
- API independence (e.g. sharing between WebNN and WebGPU)
- Browser independence
- Offline usage, including interaction with PWAs
- Cache transparency (e.g. automatic or explicit checking)

Other Issues

- Gather during breakout…

Alternatives

1. Do nothing.
2. Do the minimum.
3. Enhance existing caches.
4. Define model-aware caches.
5. Auto-expedite common models.

Next Steps

- Organize community to address problem
- Obtain consensus on solution alternative
- Further discussion of alternatives probably needed
- Do any of the alternatives need standardization

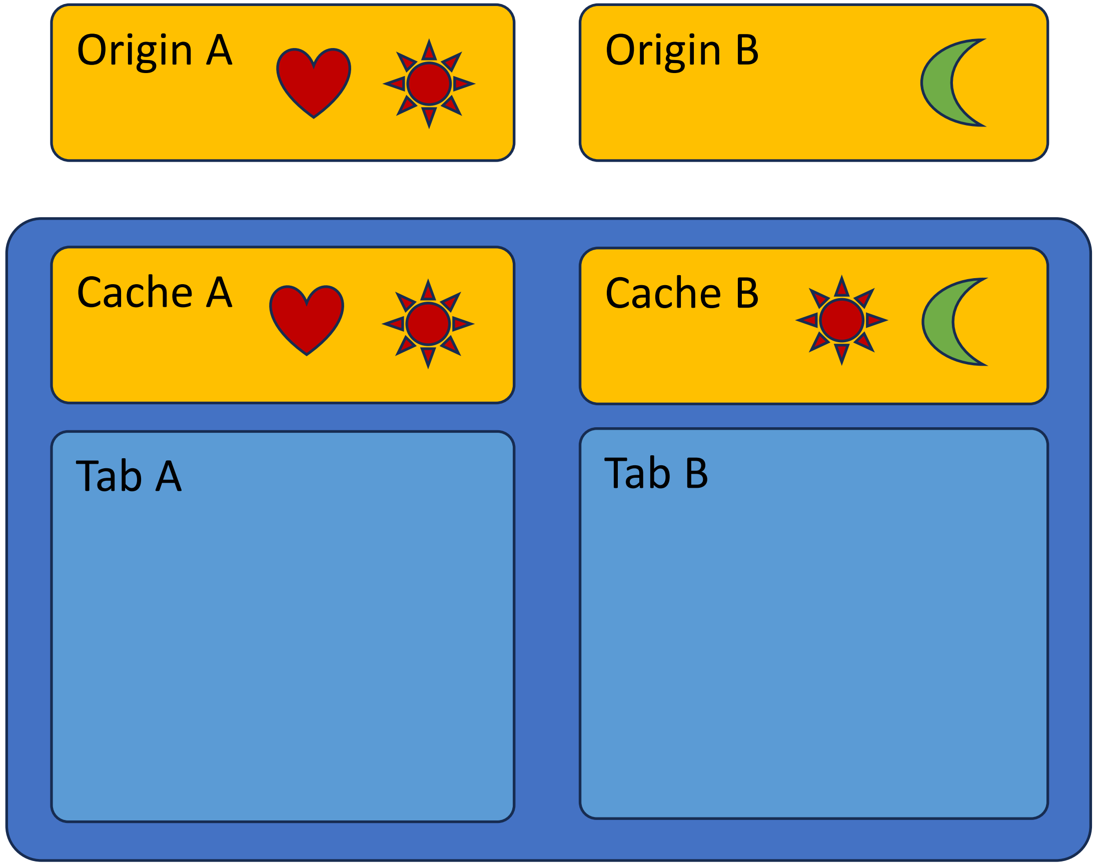
Backup

Outline

- Background:
  - Key points from stakeholder discussions
  - TPAC Breakout issues
  - Model size and download time estimates
  - Working set size and cache size estimates
  - Security and privacy considerations
- Caching: desired properties
- Alternatives
  - ***No silver bullet!***
  - Some options, but with tradeoffs, and some unfortunate complexity
- Prototype Status and Next Steps

Same-Origin and Cross-Origin Caching

Same-Origin Caches (Partitioned)



Origin A ♥ ☀

Origin B ☾

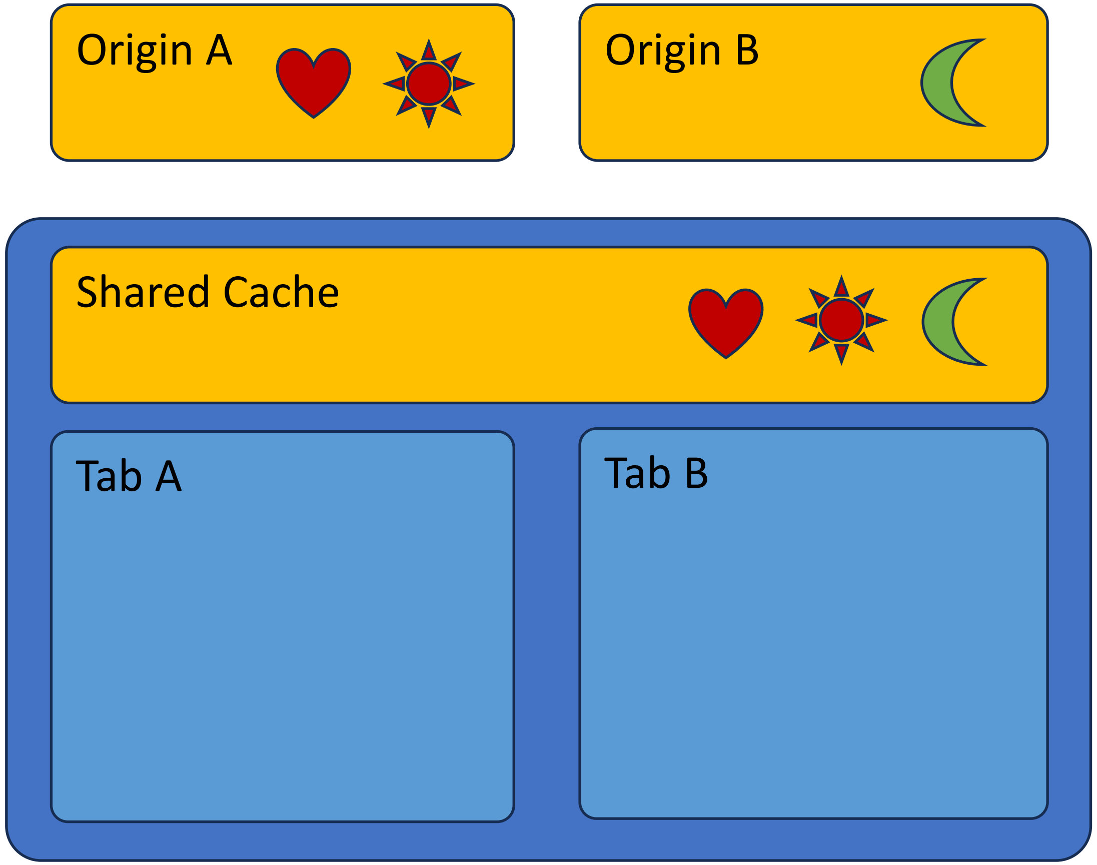
Cache A ♥ ☀

Cache B ☀ ☾

Tab A

Tab B

Cross-Origin Caching (Shared)



Origin A ♥ ☀

Origin B ☾

Shared Cache ♥ ☀ ☾

Tab A

Tab B

Possible Mitigations for Cross-Site Risks

1. Disallow use of WebNN in third-party context by default
   - *DONE: Already part of WebNN specification*
2. Generate keys (e.g. use a hash) based on actual model content
   - Avoids mass data exfiltration (block data transfers)
   - … but possibly not tracking (needs only existence checks)
3. Obfuscate cache existence check
   - At the cost of delayed access to resource (“fake miss”)
4. Limit number of built models and/or cache checks
   - Avoid use of multiple model existence checks for transferring many bits
   - May also use an “information budget” (more later)
   - May transparently shift to “same-origin-only” cache (more later)

Alternative 1: Do Nothing

1. **Use existing APIs/caches, perhaps with some extensions**
2. **Use the File System API + Background Fetch**

Alternative 1: Do Nothing

1. **Use existing APIs/caches, perhaps with some extensions**
2. **Use the File System API + Background Fetch**

- Of the available storage API options, only the File System API allows cross-site sharing
  - File System API requires “human in the loop” to provide permissions
  - Could theoretically be abused, e.g. by asking for access to non-model files
  - **Does not solve tracking/data sharing issue, just makes the user aware of the risk.**
- Other options include IndexedDB, Cache API, and the HTTP Cache, but these are all single-origin
- Cache AI models in the browser (Google)

Comment: Explicit Installation

Can ask user to "install" models explicitly

- **PWA install**
- **Extension/plugin install**

Comment: Explicit Installation

Can ask user to "install" models explicitly

- **PWA install**
- **Extension/plugin install**

- Manages user expectations that time is needed for download
  - Install process can then use background download and compilation
- If models/APIs are shared across origins and can be queried,
  - ***Does not (itself) solve fingerprinting problem***
  - In fact, shared extensions create a new major "fingerprinting surface"
  - Only option for user to preserve privacy is to not use the model
  - May still want to leverage "install moments" in combination with other alternatives

Alternative 2: Do the Minimum

Extend existing APIs/caches slightly

- **Use an extended/restricted File System API + Background Fetch**

Alternative 2: Do the Minimum

Extend existing APIs/caches slightly

- **Use an extended/restricted File System API + Background Fetch**

- Restrict the File System API to models
  - Create consistency in storage and naming
  - Avoid the user accidentally providing access to the wrong file/directory
  - Still does not address tracking/data sharing risks
- Modify Background Fetch to make it more broadly acceptable
  - Currently only supported in Chrome and Edge…
  - Address major concerns: e.g. avoid invoking Service Worker upon completion
  - Although this WOULD be useful to also compile the model "in advance"

Alternative 3: Enhance Existing Caches

1. **Enhance HTTP Cache**
    1. **Dedup contents**
    2. **Avoid fetching things it already has (needs signature check)**
    3. **“Fake miss” delay to confuse cross-site timing attacks**
2. **Implement compiled-model cache**
    1. **Dedup – if the same model is defined again, avoid recompiling it**
    2. **“Fake miss” delay to confuse cross-site timing attacks**
    3. **Compiled models may “pin” used weights in cache to avoid partial flushes**

Alternative 3: Enhance Existing Caches

1. **Enhance HTTP Cache**
   1. **Dedup contents**
   2. **Avoid fetching things it already has (needs signature check)**
   3. **“Fake miss” delay to confuse cross-site timing attacks**
2. **Implement compiled-model cache**
   1. **Dedup – if the same model is defined again, avoid recompiling it**
   2. **“Fake miss” delay to confuse cross-site timing attacks**
   3. **Compiled models may “pin” used weights in cache to avoid partial flushes**

Pros: No change to development or APIs. Relatively simple.

Cons: Not model aware. Variants or equivalent models cannot be consolidated. Models may not be treated as single entities (partial flush risk). Will not save latency on cross-site reuse. Attackers may be able to use metrics to detect fake misses. Raises risk on non-model resources.

Alternative 4: Define Model-Aware Caches

1. **Use “fake misses” to avoid redundant downloads**
2. **Progress model loads only when requesting page is inactive**
3. **Identify cache items by content-dependent hashes**
4. **Use deduplication to avoid redundant storage**

Alternative 4: Define Model-Aware Caches

1. **Use “fake misses” to avoid redundant downloads**
2. **Progress model loads only when requesting page is inactive**
3. **Identify cache items by content-dependent hashes**
4. **Use deduplication to avoid redundant storage**

- Model Caches would *behave* as if they were per-origin caches
  - Faking misses avoid redundant downloads, *but not latency*
  - Implementations can, internally, dedup to avoid redundant storage
- User override to “expedite” model load could be used
  - User then would have to accept privacy risk

Alternative 5: Auto-Expedite Common Models

- *The more common a model is, the less of a tracking risk it is*
  - Low probability models carry high information
    - …but sharing is unlikely to be useful
    - **Restrict low probability models to single-origin caching**
  - Built-in models are "certain" and carry zero information
    - 100% probability given browser+version, which is already known
    - **These would "load" immediately (would be "auto-expedited")**
    - **Would act like they are "preloaded" in the shared cache**
  - Models with in-between probabilities "budgeted"
    - Every model carries information based on log(1/p), where p is the probability
    - **Automatically expedite models in shared cache up to maximum "information budget"**
    - If information budget would be exceeded for a given probe, gracefully degrade to user expedite prompts (large models) or per-origin caches (small models)
    - ***Note that information budget check needs to happen BEFORE probe is confirmed.***

Alternative 5: Auto-Expedite Common Models

- Considerations:
  - Maintaining usage probabilities necessary…
    - How to do this fairly and in a privacy-preserving way?
  - LARGER models will have different properties than small models.
    - There are fewer of them
    - They are less likely to be “fake models” for tracking
  - Could have “model sets”
    - Reduce the number of different possible configurations and the entropy risk
    - Cache check can be for an entire model set at once
    - Can track joint probabilities/entropy for model sets
    - Joint entropy always less than sum of independent entropies

Previous Related Proposal: Privacy Budget

- Proposal by Google:
  - https://developers.google.com/privacy-sandbox/protections/privacy-budget
  - k-anonymity, entropy, differential privacy
  - When budget exceeded, certain APIs would fail
  - Proposal no longer active – ***withdrawn and archived***
- Counterargument by Mozilla:
  - https://blog.mozilla.org/en/mozilla/google-privacy-budget-analysis/
  - https://mozilla.github.io/ppa-docs/privacy-budget.pdf (details)
  - Failures due to exceeding budget can themselves be used for fingerprinting
  - Failures due to exceeding budget can cause annoying breakage
  - Correlated values make it hard to add up source entropy accurately
  - Wide ranges in probabilities mean large differences in information
    - This is the point of using something like an entropy budget in the first place, but…
    - Mozilla’s example: with fewer users, information about use of Firefox Nightly would cut into the entropy budget more than knowledge of use of Chrome… really a *fairness* issue
    - More generally: the entropy measure penalizes uncommon applications – including innovative ones.
- See also: Brave, Fingerprinting, and Privacy Budgets | Brave

Avoiding the Issues with Privacy Budgets:

Specialize “budget” concept to AI model management:

A. Failures

1. Instead of failing when the budget is exceeded, browsers can *gracefully* fall back to per-origin caches
2. The system will not “break”; worst case it may just require some load delays or redundant downloads
3. Developers should just ask for larger models first to get the most use out of the entropy budget

B. Correlation

1. It’s possible some models are often used together, e.g. are correlated
2. If A is present, and is often used with B, knowledge that B is also present will have lower information
3. Worst case: the entropy estimate will be too high, exhausting the budget early, but this is the “safe” direction
4. Better case: we can derive better joint entropy estimates for common “model sets”

C. Fairness

1. Less common models will still work under A, they just won’t go into the shared cache
2. Less common models are less likely to be shared anyway
3. Mitigations: allow the user to override entropy budget (setting to be prompted when the budget is exceeded) and/or ability to set a higher budget threshold, or perhaps even tweak the entropy assigned to a particular model; temporarily boost the expected probability of “new” models (apriori probability)

D. Fingerprinting

1. Selective budget exhaustion can itself be used to create fingerprint!
2. Mitigations: add noise to information estimates; select same-origin/shared cache with some randomness; outright reject (put in same-origin cache) very low-probability models

How to Get Model Probabilities?

- Calculating entropies of models means we need their probabilities
  - These also vary over time and need to be updated…
  - How to do this in a privacy-preserving way?
- **Option 1:** Differential privacy, e.g. add noise which is later averaged out. Unfortunately, this can mask low-probability measurements
- **Option 2:** Modular sharding, e.g. IETF PPM-DAP (Privacy-Preserving Measurement Distributed Aggregation Protocol):
  - Privacy-preserving measurement and machine learning (cloudflare.com)
  - Prio: Private, Robust, and Scalable Computation of Aggregate Statistics
  - https://datatracker.ietf.org/doc/draft-ietf-ppm-dap/

How to Get Model Probabilities?

- Still need to *distribute* results to clients in a non-centralized way
  - Could extend PPM-DAP (modular sharding) so clients are also “aggregators”
  - Clients need to query multiple servers and do final accumulation locally
- Computing model probabilities != accumulating measurements
  - PPM-DAP is designed for accumulating “measurements” of specific values
  - But we need to compute probabilities over a large set of possible models
  - If client only reports models it has, or identifies them, it defeats the purpose!
  - Options
    - A. Clients could report 0/1 for ALL possible models (expensive; 600K+ models in HF…)
    - B. Clients could randomly select models and report 0/1 (more scalable, less accurate)
      - Could use known probability distribution of models to sample space
    - C. “Surveys” could be sent out asking for data on specific models
      - Could be targeted at models for which information is needed, e.g. new models

What about non-WebNN implementations?

- Many performant implementations of AI on the Web today use WebGPU
  - Consists of compute shaders and buffers – no concept of "model"
  - Does have compiled shader cache (similar cross-site sharing issues, however)
  - Targets GPU only…
- WASM is also an implementation option
  - Targets CPU only…
- Observation: Most of the storage is in weight tensors
- **Idea:** Cache MLbuffers
  - Can't share memory across tabs, but could share "backing store"
  - Tensors stored in MLbuffers could be used by both WebNN and WebGPU
  - Potential (IF the data is stored in exactly the same way) to share tensor storage between WebNN/WASM/WebGPU implementations.

Prototype Status and Next Steps

- Implemented:
  - Node cache with hashes as keys, external Redis service for storage
  - **However:** Model cache seems to be more generally useful
- Next Steps:
  - Implement *model* cache
  - Base on Service Worker Cache, Background Fetch if possible
  - Three implementation options:
    1. Capture/replay graph building by wrapping WebNN API (shim+extension)
    2. Modify the implementation, e.g. Chromium, “under the hood” (best for performance)
    3. Cache an existing model serialization, and use a model loader
  - Write a detailed proposal document and explainer…

Backup 2

Other Security and Privacy Considerations

- **Bad:** Arbitrary key-value pairs in a shared cross-site model cache
  - Can be used for “mega-cookies” to exfiltrate data!
  - Can be also be used as trackers.
- An abuser could build a fake model
  - Embed data to be shared in the model
- Then the attacker would store the fake model in the cache.
  - Attacker can retrieve model based on key from a different site;
  - Then probe model to recover data.

NOTE: ***Service Worker Cache API or IndexedDB cannot be simply made cross-origin.***

References and Links

- Storage Partitioning (see HTTP Caches especially)
- GPU Web Privacy Considerations (shader caches)
- Felten and Schneider, Timing Attacks on Web Privacy, 2000
- Judis, Say goodbye to resource-caching across sites and domains, 2020
- CloudFlare (CDN) Origin Cache Control (can also be enabled in CDNs)
- Background Fetch – related API for large downloads.
- Cache AI models in the browser (Google) – how to use existing per-origin cache mechanisms for AI models
- https://github.com/webmachinelearning/proposals/issues/5
- https://github.com/webmachinelearning/hybrid-ai
- https://github.com/w3c/tpac2024-breakouts/issues/15
- Choose model · Issue #8 · explainers-by-googlers/prompt-api (github.com)

More References and Links

- Fingerprinting:
  - https://coveryourtracks.eff.org/
  - https://amiunique.org/
  - https://blog.amiunique.org/an-explicative-article-on-drawnapart-a-gpu-fingerprinting-technique/ (paper at https://inria.hal.science/hal-03526240/document) - can distinguish identical GPUs via WebGL
- Privacy budgets (pros, cons)
  - https://developers.google.com/privacy-sandbox/protections/privacy-budget
  - https://blog.mozilla.org/en/mozilla/google-privacy-budget-analysis/
    - https://mozilla.github.io/ppa-docs/privacy-budget.pdf (details)
  - Brave, Fingerprinting, and Privacy Budgets | Brave
- Privacy-preserving aggregation using modular arithmetic
  - Privacy-preserving measurement and machine learning (cloudflare.com)
  - Prio: Private, Robust, and Scalable Computation of Aggregate Statistics
  - https://datatracker.ietf.org/doc/draft-ietf-ppm-dap/