
WebGPT

Running GPT in vanilla Javascript



No download needed. Way cooler!
Choose chaos. Literally no libraries.

Running GPT with Pytorch



Python environments 🙄 Very cringe.
No control. Much too simple.

After six years of development, WebGPU is about to launch across most major web browsers. This is massive: web applications now have near-native access to the GPU, with the added capacity of compute shaders.

WebGPT is a vanilla JS and HTML implementation of a transformer model, intended as a proof-of-concept as well as educational resource. WebGPT has been tested to be working with models up to 500 M parameters, though could likely support far more with further testing/optimization.

Current Stats

2020 M1 Mac: 3ms/token at 5M parameters with f32 precision.

2020 M1 Mac: 30ms/token at 117M parameters with f32 precision.

2020 M1 Mac: 70ms/token at 377M parameters with f32 precision.

2020 M1 Mac: 120ms/token at 775M parameters with f32 precision.

1.5B is working but unstable, sitting around 1000ms/token due to inefficiencies.

Running WebGPT

Running WebGPT is remarkably simple, as it's just a set of HTML + JS files. Since WebGPU is still in the process of being released, you'll need to open with a compatible browser. WebGPU is currently available on Chrome v113 but the most straightforward way to ensure proper functionality is to install Chrome Canary or Edge Canary.

I've included two different models: a toy GPT-Shakespeare model (which is severely undertrained haha) and GPT-2 117M. See main.js for more information on how to run these models. If you want to import custom models, take a look at misc/conversion_scripts.

If you want to try out WebGPT, visit the demo website here KMeans.org. I'd generally recommend cloning the repo and running locally, just because loading the weights remotely is significantly slower. Note: **You'll need to use Git LFS** to download the model files, after cloning the repository.



Only 74 kilobytes in total!

Roadmap / Fixing Stupid Decisions

- ☒ Embeddings / de-embeddings on GPU.
- ☒ Initializing pipelines on every step is incredibly inefficient.
- ☒ Key-value caching.
- ☒ Reuse buffers.
- ☒ Kernel shared memory for matmul!
- ☒ Destroy buffers after use!
- ☒ Create kernel instruction classes + optimize pipeline creation.
- ☒ Fuse all kernels.
- ☒ Optimize all other kernels.
- ☒ Compute pass splitting for larger models (*maxStorageBufferBindingSize*)
- ☐ Run selection ops on GPU (topk, selection softmax)
- ☐ Attention kernel is optimized for small models, not for large models where each head having it's own matmul is more efficient.
- ☐ Investigate why attention cache isn't giving proper speed-ups.
- ☐ Make simple instructional version without special stuff.
- ☐ Optimize workgroup sizes, specifically for single row/col operations.
- ☐ Convert into a package.

☐ Write better comments + make Youtube explainer.

Acknowledgements

When I started this project I had no idea how transformers worked or how to implement them (or GPUs or matmul kernels or WebGPU or tokenization for that matter), so Andrej Karpathy's series on neural networks and building GPT from scratch were invaluable: Andrej's Youtube. I've also used some code as well from the nanoGPT repository: nanoGPT.

I copied from LatitudeGames' implementation of OpenAI's GPT-3 tokenizer in Javascript: GPT-3-Encoder.