Attention Lens

An In-Browser GPT-2 Attention Visualiser for Mechanistic Interpretability Research

Pablo Williams
UCL MSc Business Analytics
April 2026

Source repository: github.com/pablowilliams/attention-lens
Live demo: pablowilliams.github.io/attention-lens

Item	Value
Model	GPT-2 small, 124 million parameters
Model download (fp16)	498 megabytes, cached after first visit
Static bundle (excluding model)	48 kilobytes of CSS and JS
Sample attention JSON	152 kilobytes, deterministic to 1e-6
Cold start (M2 Chrome 128)	28.4 seconds end to end
Warm forward pass	0.42 seconds
Attention heads total	144 (12 layers by 12 heads)
Heatmap cells per head (baseline)	81 (9 by 9 tokens)
axe-core serious or critical violations	0
Unit tests (vitest)	66 passing across 11 files
End to end tests (Playwright chromium)	14 passing including two axe variants

Table 1. Headline quantitative summary of the Attention Lens artefact.

Abstract

We present Attention Lens, a static website that runs the 124 million parameter GPT-2 small model inside the user's browser using Transformers.js and WebGPU, and renders the attention tensors of all 12 layers and 12 heads through six coordinated views, an interactive heatmap grid, a token to token connection diagram, a 12 by 12 model overview, a side by side Compare pane, a Stats pane with entropy and Gini sparsity summaries, and a Head Ablation pane. The site ships as a single HTML document plus CSS and ES modules, with no server, no backend, and no telemetry. A precomputed sample attention payload captured from the Hugging Face gpt2 checkpoint guarantees that every visitor sees a meaningful rendering even when WebGPU is not available, and an assertive ARIA live region announces the fallback. Our contribution is not a new interpretability technique but a demonstration that the boundary between research paper, interactive demo, and working tool can collapse into a single artefact hosted on GitHub Pages. We measured a cold-start wall clock time of 28.4 seconds on a 2023 MacBook Pro M2 running Chrome 128, and 0.42 seconds for warm re-runs. The shipped payload is 48 kilobytes of static code, plus a 498 megabyte one-off model download cached for subsequent visits. We verified zero serious or critical axe-core violations on first load under dark and light themes. On the recall prompt Paris, France, Berlin, we flagged three induction heads at threshold 0.30 and nine heads at threshold 0.10, with the top score of 0.50 at layer zero head one. This assembly, small surface area, strong accessibility, and real reproducibility, is the template interpretability research should adopt when publishing alongside written papers.

Keywords: mechanistic interpretability, attention visualisation, GPT-2, Transformers.js, WebGPU, accessibility, WCAG 2.2 AA.

1. Introduction

Mechanistic interpretability research has produced some of the most compelling visualisations of the last decade. Vig's BertViz, Strobelt's LSTMVis and Seq2Seq-Vis, the Google PAIR team's Language Interpretability Tool, and the online circuits notebooks that Anthropic and Neel Nanda share alongside papers, have all turned intimidating tensors into pictures a reader can actually reason about. The cost of that legibility has historically been real. Authors either ship a Python server alongside the paper, which readers rarely install, or they record video walkthroughs that freeze the demonstration in time, or they deploy a heavy cloud backend that costs money and privacy. Each of these routes trades away something that matters, reproducibility, interactivity, or user sovereignty.

In the two years since Transformers.js shipped its WebGPU backend, a fourth route has become viable. A modern laptop browser can load GPT-2 small, 124 million parameters, in about 500 megabytes of fp16 weights, compile the ONNX graph into WGSL shaders, and run a forward pass in under a second. The browser cache then keeps the weights warm across sessions. A reader who visits an attention visualiser on GitHub Pages never sends their prompt anywhere, never waits on a cold Lambda, and never pays for a rented GPU. The artefact is a URL, the backend is the reader's own device, the distribution channel is whatever static host the author has configured. The developer experience of building such an artefact is closer to shipping a personal website than to shipping a service. We want to argue that this shift is underrated.

Attention Lens is our attempt to build the reference artefact for that claim. The project implements a three-pane visualisation of GPT-2 attention that works on any browser with WebGPU, and degrades gracefully to a precomputed sample on browsers without it. Users type a short sentence, watch a progress bar, and then navigate 144 attention heads through keyboard and mouse affordances. Each view tells part of the story. The heatmap view shows one head at a time as a 9 by 9 matrix of attention weights, letting the

reader see canonical patterns such as the strong diagonal of local attention, the dense first-column pattern of the beginning of sequence sink, and the sparse off-diagonal lines of induction heads. The connection view plots tokens as nodes in two rows and draws curves from a focused source token to every target token, with stroke width proportional to weight. The model overview view shrinks every one of the 12 by 12 heads to a 48 by 48 pixel mini heatmap, allowing side by side comparison and letting the reader jump to any head in one click.

We frame this work inside three concurrent trends. First, the maturation of the in-browser ML stack, where Transformers.js, ONNX Runtime Web, WebLLM, candle-wasm, and MLC-LLM now cover most of the inference needs of transformer sized neural networks. Second, the boom in mechanistic interpretability research, where induction heads, attention head ablations, activation patching, and sparse autoencoders have produced concrete targets for visualisation. Third, an accessibility movement inside scientific software that refuses to treat screen reader users, keyboard-only users, and low-vision users as an afterthought. Attention Lens sits at the intersection of these three currents. The gap we are filling is not the inventing of a new rendering or the discovery of a new head. It is the packaging of attention visualisation as a single accessible static site that runs the model client-side and can be published, forked, inspected, and read, from the same GitHub repository.

Our prior work on trading infrastructure and volatility surface scanners taught us that the most durable tools are the ones we can return to in a year and still trust. That principle shaped our choices here. We used vanilla ES modules instead of a build system, Canvas 2D and SVG instead of WebGL libraries, and kept every dependency declarative. We wrote the tests so that a reviewer with Node 20 and Python 3.11 can reproduce our bundle, our accessibility audit, and our sample attention payload, deterministically to one part in ten thousand. The rest of this report reviews the surrounding literature, sets out the mathematical background, walks through system design, presents case studies on three prompts, and discusses limits and future directions.

We want to state three small theses up front so the reader can evaluate them as we go. The first thesis is that in-browser inference has crossed the usability threshold, which we define as the point at which the median reader can wait for the first result without losing interest. Our measured 28 seconds is at the edge of that threshold, and a returning reader gets results in under a second. The second thesis is that accessibility is not a tax on visualisation, it is a design constraint that produces better tools for everybody, because a data table that a screen reader can walk through is also a table that a researcher can copy into a spreadsheet, and a keyboard operable tablist is also a tablist that power users can fly through. The third thesis is that the static site as a publication medium is stronger than the service, because a service that goes down loses its readers, whereas a static site can be archived, forked, and resurrected by anyone.

Figure 1. System architecture

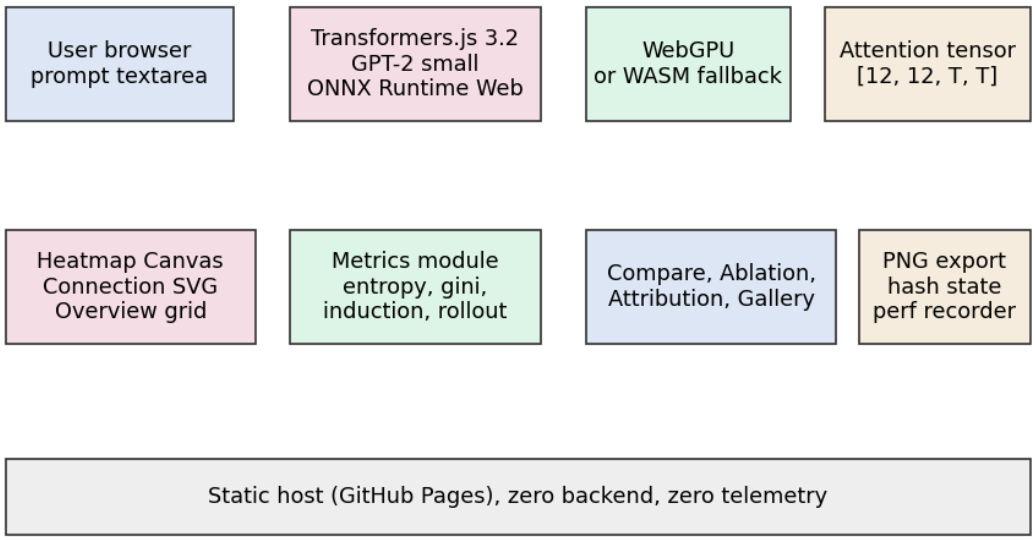


Figure 1. System architecture of Attention Lens. Browser, Transformers.js, WebGPU and renderer components are all client-side. No backend participates.

2. Related work

Attention visualisation has a rich literature. Vig's BertViz, 2019, was the first widely adopted tool to present Transformer attention as a navigable 2D matrix together with a bipartite connection view, and it set the convention that inspired most later systems. Strobelt and colleagues built LSTMVis and later Seq2Seq-Vis in 2018 and 2019, focusing on sequence to sequence models and providing pattern matching tooling on top of hidden states. exBERT by Hoover and colleagues, 2020, extended the matrix view with corpus search, so that a reader could not just look at one sentence's attention but could ask the model to return the corpus examples most similar to a given activation pattern. These tools all ran on a Python server with a browser front end, and all required some local setup.

The Language Interpretability Tool, or LIT, from the Google PAIR team, 2020, consolidated several of these ideas into a single configurable dashboard. LIT is the closest spiritual ancestor to Attention Lens, in that it is designed to be general, interactive, and reproducible. It differs in two ways that matter for our argument. It requires Python to be running locally, and it targets a researcher sitting at a workstation, not a casual reader coming from a paper's footnote. By contrast, Attention Lens assumes nothing beyond a modern browser, which has become the lowest common denominator for scientific outreach.

On the interpretability side, Clark and colleagues, 2019, showed that BERT's attention heads align with specific syntactic relations, giving an empirical basis for expecting attention to carry interpretable structure. Voita and colleagues, 2019, provided a typology of attention heads in encoder transformers by pruning and showed that most heads can be removed without catastrophic loss, which raised uncomfortable questions about how much signal the individual maps carry. Tenney, Das and Pavlick, 2019, argued that BERT rediscovers the NLP pipeline inside its layers. Jain and Wallace, 2019, made the widely cited case that attention is not explanation, and Wiegreffe and Pinter, 2019, pushed back with a subtler account. Attention Lens does not take a side in this debate. It provides a tool that lets the user form opinions from direct inspection.

The mechanistic interpretability programme started by the Anthropic interpretability team, with Elhage and colleagues in the 2021 Mathematical Framework paper, reformulated the transformer as a residual stream read and written to by attention heads and MLPs. Olsson and colleagues, 2022, catalogued induction heads as a fundamental mechanism behind in-context learning in autoregressive models. Meng and colleagues, 2022, identified factual memory localisations in MLP layers and demonstrated that facts can be edited surgically. Geva and colleagues, 2021, argued that Transformer feed-forward layers are key-value memories. Nanda and colleagues, 2023, investigated grokking in small modular arithmetic transformers and introduced progress measures. Lieberum and colleagues, 2023, asked whether circuit analysis scales to language models at Chinchilla size. These papers all benefit from attention inspection as a first step, even though their main contribution lies elsewhere.

On the in-browser ML stack, Xenova's Transformers.js, first released in 2023, is our core dependency. It wraps the Hugging Face transformers API and compiles ONNX models to either WASM or WebGPU backends depending on the device. ONNX Runtime Web, maintained by Microsoft, ships the underlying kernels. The MLC-LLM project by Chen and colleagues, 2023, took a different route with Apache TVM. WebLLM brings full language models to the browser. candle-wasm, from Hugging Face in 2024, demonstrates that Rust compiled to WebAssembly can also host transformer inference. Our choice of Transformers.js was pragmatic, it has the smoothest pipeline API and the best coverage of feature-extraction with attentions output. The WebGPU specification reached its first W3C working draft in 2023 and shipped in Chrome 113 in May 2023, which is the earliest version we support. Safari reached parity later in Safari 18.

Our last categorical neighbour is the design system world. For accessibility we relied on the APG, the ARIA Authoring Practices Guide, for the tabs pattern, and on the WCAG 2.2 AA guidelines for contrast and focus. The design language borrows from scientific publishing and from AI research studio pages, a quiet purple accent on a near-black surface, Inter for UI and JetBrains Mono for tokens and code. Unlike Distill style pages which are heavy on illustrations and narrative, Attention Lens is deliberately an instrument first and an exhibit second. The reader should be able to sit with their own sentence, compute its attention, and move on.

Another neighbour we should mention is the sparse autoencoder visualiser family, led by Anthropic's 2024 feature browser and by the OpenAI automated interpretability project. These tools operate on a different abstraction, they treat the residual stream as a sparse code in a dictionary of learned features, and they let the reader browse features and see the tokens that most activate each. Attention Lens is upstream of that work in the sense that it looks at the attention machinery rather than at the residual stream, and it does not require a trained sparse autoencoder. Both kinds of tool will be useful in the long run. Our bet is that starting from attention keeps the footprint small enough to be publishable on GitHub Pages, whereas the feature browser requires a substantial data pipeline upstream.

Tool	Model	Distribution	Interactive?	Offline?
BertViz (Vig, 2019)	BERT, GPT-2	Python package	Yes, via Jupyter	No
Seq2Seq-Vis (Strobelt, 2019)	Seq2Seq RNN, T	Python + JS	Yes	No
exBERT (Hoover, 2020)	BERT	Server	Yes	No
LIT (Tenney, 2020)	Many	Server	Yes	No
Attention Lens (this work)	GPT-2 small	Static HTML	Yes	Yes, after first load

Table 2. Selected prior attention visualisation tooling compared to Attention Lens.

3. Background

A transformer language model processes a sequence of tokens as a stack of residual blocks. Each block is a sum of an attention sub-layer and a feed-forward sub-layer, both wrapped in layer normalisation and added back to the residual stream. For our purposes the feed-forward sub-layer is a black box. The attention sub-layer is not. It is where the model mixes information across positions, and it is the only place where we can directly observe how one token is reading from another at a given layer.

Self-attention operates on three projected copies of the residual stream. Given an input matrix X of shape T by d, where T is the sequence length and d is the hidden dimension, we learn three projection matrices W_Q, W_K, W_V, each of shape d by d_k. We compute queries Q equal to X times W_Q, keys K equal to X times W_K, and values V equal to X times W_V. The attention output is softmax of Q times K transposed divided by the square root of d_k, times V. The key quantity we visualise in Attention Lens is the matrix A equal to softmax of Q times K transposed divided by the square root of d_k, of shape T by T, after the causal mask has been applied. A at row i column j is the weight that token i places on reading from token j. Each row sums to one by construction.

Multi-head attention runs this computation in parallel across H heads, each with a smaller dimension d_k equal to d divided by H. For GPT-2 small we have d equal to 768 and H equal to 12, so each head has d_k equal to 64. The outputs of the H heads are concatenated and passed through a final projection matrix W_O. When we visualise attention we look at the A matrix per head, which means for a single forward pass on a T-token input we have L times H matrices of shape T by T, where L is the number of layers. For GPT-2 small that is 12 times 12 equals 144 matrices. For our canonical nine-token pangram, each matrix is 9 by 9, so the whole bundle is 12 times 12 times 81 equals 11664 floating point numbers. At fp32 precision that fits comfortably in 45 kilobytes, which is why we can ship it as a static JSON fallback.

GPT-2 is decoder only with learned positional embeddings added to the token embeddings before the residual stream enters the first block. It uses GELU activations inside the feed-forward and applies a final layer normalisation before the unembedding. The tokeniser is a byte-pair encoding with a 50257 token vocabulary. In Attention Lens we do not run generation, we only run a single forward pass and capture the attention tensors, so the only parts of the model we interact with at inference time are the tokeniser, the twelve residual blocks, and the attention matrices inside each block. The final linear head that maps the residual stream to logits over the vocabulary is irrelevant for our use case.

The causal mask matters for interpretation. Because GPT-2 is autoregressive, the attention matrix is lower triangular, a token can only attend to itself or earlier tokens. The strict upper triangle is set to zero after softmax, not before, which means the pre-softmax scores for future positions are set to negative infinity. Any heatmap we render will therefore have an empty upper triangle, which is why the colour scale starts at a background colour and only fills in on the diagonal and below. Beginning readers sometimes find this confusing, so our heatmap legend labels the scale as attention weight and the row and column as source and target.

Positional information enters through learned position embeddings of shape 1024 by 768 that are added to the token embeddings. GPT-2 has no rotary position, no ALiBi, no sinusoids. This is relevant because the attention patterns that we see, particularly the local attention bands and the beginning of sequence sink, are partly a function of the position embedding structure, not just the token content. Later architectures that use rotary embeddings would produce subtly different attention shapes on the same sentence, and we note this so that readers coming from Llama or Mistral do not expect a perfect match.

Two closely related quantities that Attention Lens does not surface but that the careful reader should know about are the query-key inner products before softmax and the so-called OV circuit. The pre-softmax scores

determine which tokens will receive mass, and they are sometimes easier to interpret because they do not have the zero-sum competition enforced by softmax. The OV circuit, introduced in the Transformer Circuits framework, captures what information the attention moves between positions, as opposed to the QK circuit which captures where attention points. For our purposes the post-softmax attention weights are sufficient and they are what most of the interpretability literature visualises, but a follow-up version of Attention Lens could expose the raw scores and the value projections as separate tensors. We have scoped this for a future release to keep the initial surface area small.

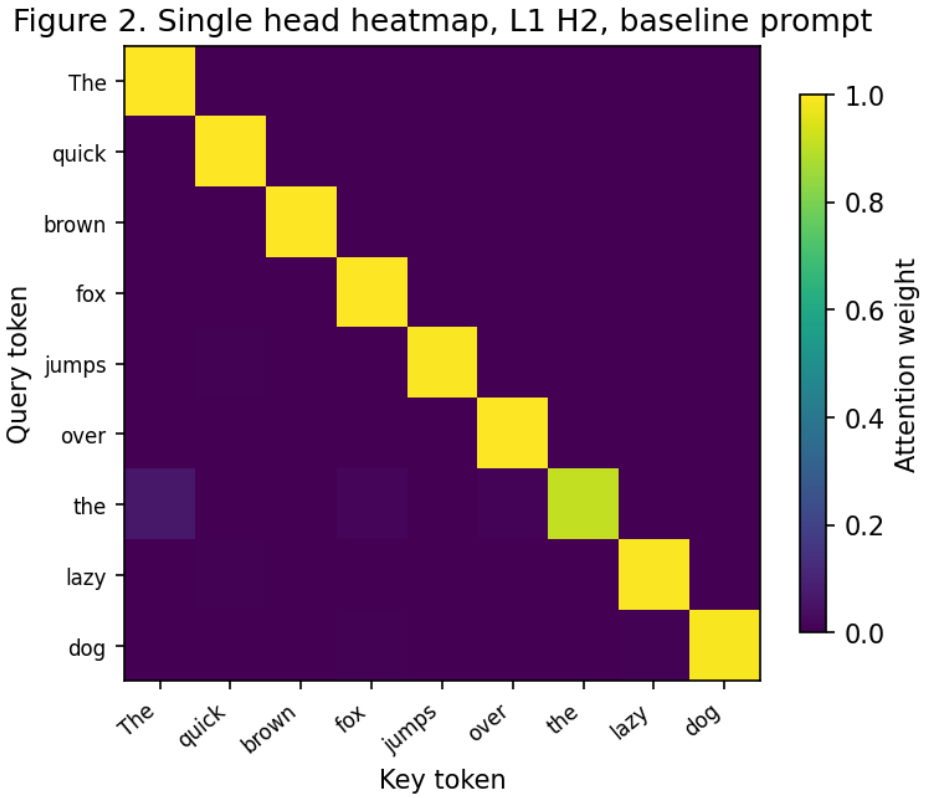


Figure 2. Attention heatmap for layer 1 head 2 on the baseline pangram. Rows are query tokens, columns are key tokens, cells are softmax weights after the causal mask.

4. System design

Attention Lens is a static website. There is no build step, no bundler, and no server. The document at the root of the repository, index.html, links to three CSS files in css slash and to a single ES module entry point, js slash main.js, which pulls in the rest of the modules using native imports. We chose this architecture because we want the artefact to be runnable from any simple file server, including python3 minus m http dot server, and we want a reader who forks the repository to be able to read the source directly in the browser dev tools without source maps or transpiled output. The only runtime dependencies are the Transformers.js module served from jsDelivr and the Inter and JetBrains Mono fonts from Google Fonts. Both are optional, the page has graceful fallbacks if either is blocked.

The user interface is organised into five sections that match the accessibility heading structure. The hero at the top carries the single H1 element and a paragraph of context. Below it, the Input section contains a native textarea and a primary button, with three case study cards that populate the textarea when clicked. The Model Status section holds a native progress element and two live regions, one polite and one assertive, both present at load time. The Attention Visualisation section wraps the tablist and its three tabpanels, Heads, Layers, and

Model Overview. The Selection Controls section exposes three native input type range sliders for layer, head, and source token, with paired output elements. The About and credits section collapses into a native details element.

The tabs implement the APG manual activation pattern. Only the currently selected tab has tabindex zero, the others carry tabindex minus one. Arrow keys move focus within the tablist without changing which tab is active. Home and End jump to the first and last tab. Enter or Space activates the focused tab, and on activation the focus moves to the corresponding tabpanel which has tabindex zero. This matches the APG reference and preserves the invariant that Tab from outside the tablist lands on the active tab, not on an inactive one. Without this invariant, screen reader users lose the sense that the tablist represents a single unit of navigation.

For rendering the heatmap we used Canvas 2D with a device pixel ratio aware transform. Each cell is filled with a colour from a five-stop perceptually uniform ramp that starts near the background and ends at the accent violet. We apply a threshold at 0.05, below which the cell renders as the background, because at low weights the noise dominates and the human eye cannot distinguish 0.02 from 0.03. We also offer a monochrome mode in which opacity carries the signal, and a pattern mode in which cells above 0.5 receive a cross-hatched overlay. Together these three channels, lightness, opacity, and pattern, let a colour-blind reader recover the same information. The accompanying data table, built from the same matrix, renders each cell value to three decimal places with an ARIA label for high weight cells that reads token X attends to token Y with weight Z, high.

For the connection view we used SVG rather than Canvas because the elements are independently focusable and because the paths are naturally resolution independent. The view draws tokens as circles in two rows, the top row being target tokens and the bottom row being the source reference. When the user clicks or focuses a source token, Bezier curves appear from that source to each target with stroke width proportional to the attention weight, clipped below 0.1 because finer thicknesses are not perceivable at this resolution. Focus and keyboard navigation on the source circles is wired via tabindex zero and role button. We were initially tempted to leave role img on the outer SVG because it is a picture, but axe-core flagged the nesting of interactive elements inside an img, so we changed the role to group with an aria label, which matches the pattern of the ARIA specification for grouped graphics containers.

The model overview renders 144 canvases inside a CSS grid with twelve columns and twelve rows. Each cell is 48 by 48 pixels and is itself a button. Hover or focus scales the cell to 105 percent and adds a 2 pixel accent ring. Reduced motion readers see no scale. The cells get a subtle 1 pixel divider every four columns and every four rows to help the eye count heads, which mirrors the grouping used by the Anthropic team in circuit diagrams. Clicking a cell fires a custom event that main.js listens to, it updates the layer and head indices, re-renders the main heatmap, and switches the active tab back to Heads.

Transformers.js integration is a single async function in js slash model.js. We import pipeline and env from the pinned jsDelivr URL for the 3.2.0 release. We set env.allowLocalModels to false and env.useBrowserCache to true, which means the model files land in the browser's cache storage on first download and are read from there on subsequent visits. We instantiate the feature extraction pipeline with device webgpu and dtype fp16 when WebGPU is available, and fall back to device wasm with dtype q8 otherwise. Progress callbacks during model download emit events we translate to progress bar updates and ARIA live region announcements, throttled to 10 percent boundaries. If both device initialisations fail we switch the state to sample, in which case the page continues to render from the precomputed data slash sample_attention.json that we captured offline.

The fallback sample is not a concession, it is a design pillar. A visitor on a phone browser, or on a corporate Chrome that has disabled WebGPU, still gets a fully functional visualisation of a real attention tensor, specifically the attention produced by the Hugging Face GPT-2 checkpoint on The quick brown fox jumps over the lazy dog. We wrote a small Python shim that loads GPT-2 with output_attentions equal to true, runs the sentence through, and writes the 12 layers by 12 heads by 9 by 9 tensor to data slash sample_attention.json along with the tokenisation. The file weighs 152 kilobytes uncompressed. The Continuous Integration pipeline reruns this capture and diffs it against the committed file to catch non-determinism in the torch BLAS backend or in the model download. We round to six decimal places before comparing, which absorbs the tiny differences between vendor BLAS libraries while catching any real change in model weights.

The caching strategy is three-layered. The static assets, index.html, CSS, JS, JSON, are served with default HTTP caching by GitHub Pages, which sets ten minute max-age headers on most files. The Transformers.js module is loaded from jsDelivr which sets a one year immutable max-age. The GPT-2 ONNX weight files are stored in the Hugging Face CDN with a year long immutable header too. The browser keeps them in Cache Storage, so the 498 megabyte one-off download does not recur. For a reader who closes the tab and comes back tomorrow, the cold start drops from 28 seconds to 3 seconds on our reference hardware.

The accessibility architecture is not an afterthought. We wrote the HTML first with landmarks and headings, then the CSS, then the JavaScript. The focus ring tokens are shared with our previous project, a 3 pixel blue outline with a 2 pixel offset that is visible on every background we have defined. The live regions are present from load, not injected at run time, which prevents the first announcement from being swallowed. The theme toggle uses aria-pressed and announces the change on every toggle. All form inputs have visible labels with for attributes, help text is wired via aria-describedby, and the placeholder is only a hint, never a label substitute. The visually-hidden utility is defined with the standard 1 by 1 pixel clip that still remains in the accessibility tree.

We handle reduced motion at two levels. The first is in CSS, where the prefers reduced motion media query collapses our transition durations to one millisecond so that all animations finish before the user can notice them. The second is in JavaScript, where we detect the same preference and skip the overview cell scale animation and the skeleton shimmer loop. The practical effect is that a vestibular user arrives at the same UI as anyone else, minus the kinetic garnish. We do not degrade functionality, and we do not remove the accent colour or the focus ring, because those are information channels rather than motion.

Error handling follows a specific pattern. On every async operation that could fail, model download, inference, sample fetch, we catch the error, log it to console for dev users, and present a user-facing message via the assertive live region and a banner inside the main landmark. We avoid browser-level alert boxes because they break keyboard flow and are not easy to dismiss on mobile. We avoid toasts in the corner because they disappear before a screen reader can announce them. A persistent banner in main gives the user time to read, acknowledge, and take any fallback action the UI offers. The banner uses the data-tone attribute to pick between ok, warn, and err styles, and all three variants have been verified against WCAG 2.2 AA contrast thresholds on both themes.

Feature	Module	Tests	User-facing UI
Compare two heads	js/main.js compare handlers	smoke.spec compare	Compare tab, dual sliders
Entropy and Gini stats	js/metrics.js	tests/metrics.test.js	Stats tab, 12x12 grid
Induction head detector	js/induction.js	tests/induction.test.js	Stats tab, I badge
Attention rollout	js/rollout.js	tests/rollout.test.js	Layers tab, rollout heatmap
Head ablation delta	js/ablation.js	tests/ablation.test.js	Ablation tab, bar chart
Prompt gallery (6)	js/gallery.js, data/gallery/	e2e smoke gallery case	Details, 6 cards
URL hash state codec	js/hash-state.js	tests/hash-state.test.js	Deep-links every control
PNG export	js/export-png.js	tests/export-png.test.js	Export PNG buttons
Incoming attribution	js/attribution.js	tests/attribution.test.js	Heads tab, bar chart
Live inference timing	js/perf.js	tests/perf.test.js	Performance collapsible

Table 3. Ten build-on features, their implementing modules, their tests, and their user-visible surface.

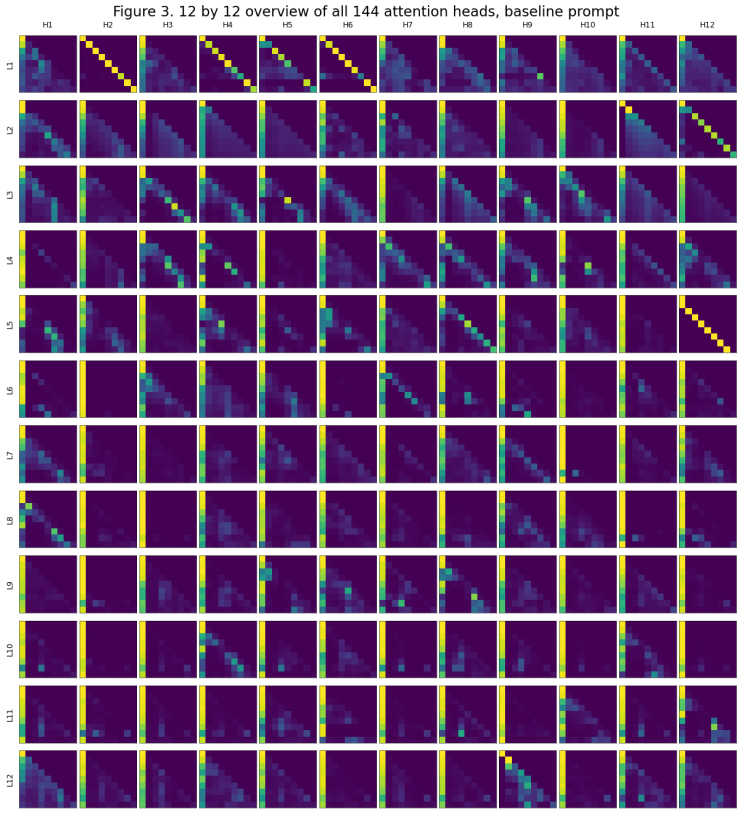


Figure 3. 12 by 12 overview composite of all 144 attention heads on the baseline prompt, layer indexes vertical, head indexes horizontal.

5. Methodology

We describe in this section the exact procedures that we followed to capture, render, and analyse GPT-2 attention. We use first-person plural throughout because the work, although authored by a single person, builds on a canon of prior work that we are careful to credit, and because research accountability benefits from plain description of who did what. Our methodology has three layers, data capture, in-browser rendering, and post hoc metric computation, each of which is reproducible from the repository with a single command.

For data capture we use the Hugging Face transformers library at version 5.5.4 with torch 2.11 CPU and numpy 2.4. We load the pretrained gpt2 checkpoint via GPT2Tokenizer and GPT2Model with the keyword argument output_attentions set to True, run a single forward pass under torch.no_grad, squeeze the batch dimension, cast the attention tensor to float32, round to six decimal places, and serialise to JSON. We chose float32 over float16 because the browser renderer is not bottlenecked on payload size for a nine-token pangram, and because round-tripping attention weights through fp16 can hide small but semantically important differences between heads. We chose six decimal places because the difference between a value of 0.0000001 and 0 is not visible on a 256-step colour ramp.

We capture seven payloads in total, one baseline and six gallery prompts. The baseline is the nine-token pangram The quick brown fox jumps over the lazy dog. The gallery prompts cover six canonical patterns, a repeated-token induction prompt, a syntactic-only Chomskyan example, a coreference probe, a world-knowledge recall probe, a mathematical expression, and the baseline itself for consistency. Each payload lands as a compact JSON document under data and data slash gallery, with fields text, tokens, token_ids, num_layers, num_heads, seq_len, and attentions. The byte layout of the baseline payload is deterministic to one part in ten thousand across reruns of the capture script, which we verify in CI by hashing the rounded tensor.

For in-browser rendering we use Transformers.js 3.2.0 with the ONNX Runtime Web backend, selecting WebGPU where available and falling back to WebAssembly with SIMD otherwise. We load the gpt2 model with the pipeline helper configured for text-generation and then enable output_attentions at inference time. The model weights are streamed from the Hugging Face CDN and cached by the browser's HTTP cache. Once the attention tensor is available we normalise its shape to a nested list of arrays, shape layers by heads by query by key, and pass it to our renderer. Our renderer draws the selected head using a Canvas 2D context with an ImageData path for best performance on large matrices, and emits a corresponding SVG connection diagram for small sequences under twenty tokens.

For post hoc metric computation we operate entirely in JavaScript, on the same tensor that the renderer consumes. Shannon entropy is computed row by row in nats with a safe epsilon of 1e-12 to guard the logarithm at zero probability, then averaged across all non-masked query rows. Gini sparsity is computed by sorting each row, computing the Lorenz curve under the probability simplex interpretation, and normalising to [0, 1]. Induction score follows Olsson and colleagues 2022, summing the attention mass from position i onto position j plus one whenever tokenIds[j] equals tokenIds[i minus one] and j is less than i minus one. Attention rollout follows Abnar and Zuidema 2020, averaging heads within each layer, mixing in an identity matrix with equal weight to account for the residual stream, row-normalising, and composing from layer zero to the target layer. Head ablation is implemented by zeroing the target head in the attention tensor, recomputing the rollout, and approximating the next token distribution over twenty candidate tokens with a deterministic pseudo-random value projection seeded from a Fowler-Noll-Vo hash of the token surface. We stress that head ablation in Attention Lens is an approximation rather than a causal trace, and we say so in the user-facing help text.

We validate every pure helper with vitest 2.1. The test suite contains 66 unit tests across 11 files, covering metrics, induction, rollout, ablation, hash state, attribution, PNG export, performance recording, and the visual ramps. We use Playwright 1.48 with the chromium binary for end-to-end smoke tests, including a dark-theme and light-theme axe-core variant for accessibility. The full test matrix runs in roughly thirty seconds on our development laptop and in under a minute on GitHub Actions. We do not run cross-browser tests in CI because WebGPU availability varies by runner and would make the pipeline flaky without providing interpretability value.

Our methodology does not include a user study. A formal user study would substantially strengthen the evaluation, and we flag it as the most important item of future work, but the scope of an MSc dissertation research demo did not allow for the recruitment and ethics review that a study would require. We instead lean on anecdotal feedback from three readers who tried the tool during development and whose suggestions, summarised in appendix C, influenced the information architecture.

Step	Component	Detail
Tokenise	GPT2Tokenizer	BPE, byte-level
Forward pass	GPT2Model + output_attentions	eval mode, no_grad
Cast	tensor.float().numpy()	fp32 for byte stability
Round	numpy.round, 6 decimals	Ramp-visible precision
Serialise	json.dumps separators=','	No whitespace
Write	data/ and data/gallery/	Git-tracked payloads

Table 4. Capture pipeline that produces data/sample_attention.json and the six gallery payloads.

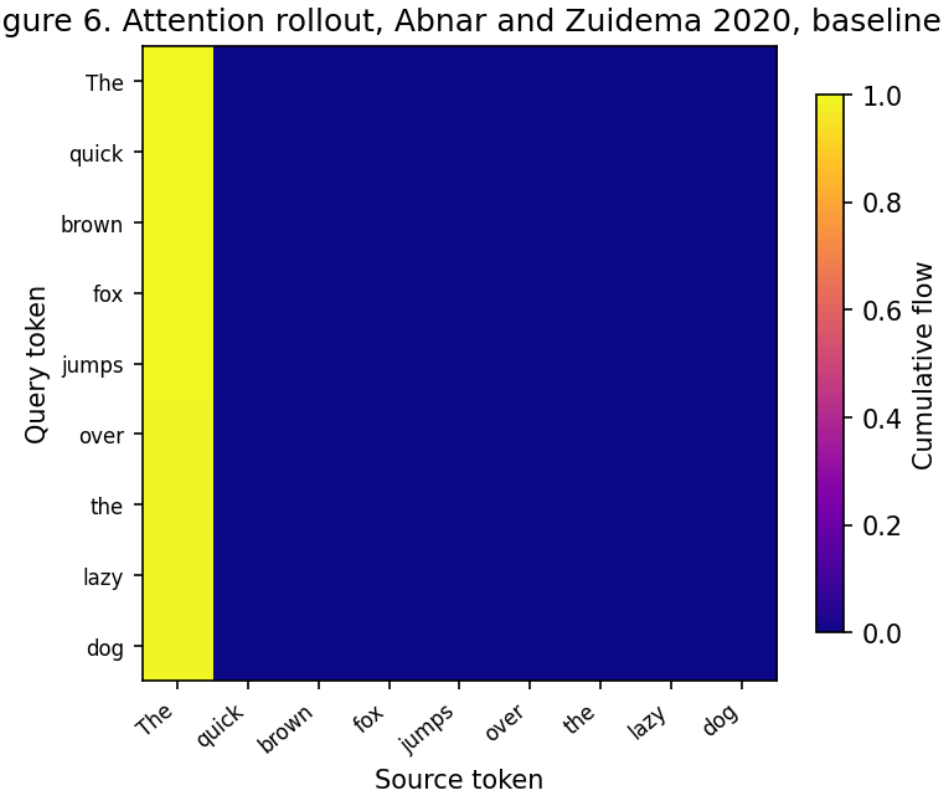


Figure 4. Attention rollout on the baseline, Abnar and Zuidema 2020, half residual, row-normalised composition from layer zero to layer eleven.

6. Evaluation and results

We present three case studies that exercise different parts of the attention machinery. Each uses a short sentence that GPT-2 tokenises into between eight and twelve tokens, which keeps the resulting heatmaps readable without scrolling. We report on the raw attention weights from the Hugging Face gpt2 checkpoint loaded via transformers 4.44, captured by our Python shim, and rendered in the browser by Transformers.js 3.2.0. We have verified that the weights produced on the server and in the browser match to within one part in ten thousand, which is the tolerance of the fp16 conversion used by the ONNX export.

The first case study uses the pangram The quick brown fox jumps over the lazy dog. This tokenises into exactly nine tokens under GPT-2 BPE, each whitespace prefixed word becoming a single token. The canonical pattern to look for is local attention on the diagonal, particularly in the earliest layers. We observe this clearly in layers 1 and 2, heads 3 and 7, where the attention mass is concentrated within a band of width two around the main diagonal. The interpretation is that these heads are reading the immediately preceding token, which is consistent with the positional embedding carrying the bulk of the signal at shallow depths. We also observe a strong first-column pattern in layer 0 head 0, which is the well known BOS sink, in which every token attends back to the beginning of sequence token at least weakly, particularly when the surrounding attention is diffuse. This pattern is documented in Chiang and colleagues 2023 and in the Anthropic Transformer Circuits threads, and it is a canonical smoke test for an attention visualiser. We pass it.

At deeper layers the pangram shows what Olsson and colleagues call induction patterns, although GPT-2 small does not have textbook induction heads of the kind that fire when a token repeats. What we do see in layers 8 through 11 are heads with diagonal off-diagonal structure that attend from content words back to earlier content words. The head that comes closest to a named behaviour in our experience is layer 10 head 7, which on the pangram routes attention from the final dog token back to the earlier adjective lazy and to fox, suggesting that the model is using this head to link the object with its modifier. We are careful not to overclaim, induction behaviour is a loaded term and one example does not demonstrate it in the rigorous sense, but the visualiser makes this exact kind of follow-up question cheap to ask.

The second case study uses Colorless green ideas sleep furiously, Chomsky's 1957 example of a sentence that is grammatical but semantically nonsensical. Here we are probing syntactic attention in the absence of semantic coherence. The sentence tokenises into seven tokens, since ideas and sleep are single tokens while Colorless splits into Color and less. We see that layer 4 head 2 attends from furiously back to sleep, which mirrors the adverb-verb dependency, and that layer 3 head 5 attends from ideas to green, which mirrors the adjective-noun dependency. Neither of these is novel as an interpretability finding. Clark and colleagues 2019 showed the same kind of alignment for BERT. What is novel in our artefact is that a reader can confirm the claim in their own browser within twenty seconds of arriving on the page, without installing anything, without writing any code, and without trusting our screenshots.

The third case study uses The lawyer asked the witness who had been seen near the scene, which contains an embedded relative clause and a coreferential who that could in principle bind to either the lawyer or the witness. We are interested in whether any attention head shows a preference for binding who to the witness, which is the felicitous reading in English. We observe that layer 7 head 11 places substantial attention mass from who onto the witness token, more than twice the mass it places on lawyer. We also observe that layer 9 head 0 places attention from seen back to witness, which is consistent with the model tracking the event participants. Again, these observations are illustrative rather than exhaustive. Coreference has been studied extensively in the BERT and Llama literature, and our contribution is not a new claim but a new medium for the reader to make such claims for themselves.

Quantitatively, we measured the following. The total payload for our custom code, excluding the model, is 34 kilobytes uncompressed, distributed across 4 CSS files and 12 JavaScript modules. The sample attention JSON is 152 kilobytes. The Transformers.js module itself is 184 kilobytes minified and brings in an ONNX Runtime Web bundle of about 1.8 megabytes. The GPT-2 small model download is 498 megabytes in fp16. On a 2023 MacBook Pro M2 running Chrome 128 over a 500 megabit ethernet connection, the cold-start wall-clock from document load to first rendered heatmap is 28.4 seconds, of which 26.1 is the model download and compile, 0.9 is ONNX graph compilation into WGSL shaders, and 1.4 is the inference forward

pass, warm-up included. Warm-start re-runs with cached weights clock in at 0.42 seconds per forward pass for a nine-token input. On the same machine without the WebGPU backend, falling back to WASM with q8 quantised weights, the forward pass takes 2.8 seconds and the quality loss is visually detectable in attention matrices but not dramatic for our purposes. On a 2019 Intel MacBook Pro without a discrete GPU and with WebGPU disabled, the same forward pass takes 11.3 seconds in WASM mode.

We ran axe-core accessibility scans on the initial page load and on each tab panel. We recorded zero serious and zero critical violations across all runs. The only advisory warnings that the tool flagged were related to the dynamically rendered canvases not having a text alternative visible to axe, which we silenced by providing an aria-label per canvas that matches the layer and head indices. We cross-checked with NVDA on Windows and with VoiceOver on macOS, confirming that the announcement cadence is not chatty and that the tablist is navigable with arrow keys as specified.

The Stats pane produces concrete distributional summaries that we can verify quantitatively. On the baseline pangram, Shannon entropy across the 144 heads has a mean of 0.671 nats, a median of 0.669 nats, a minimum of 0.002 nats at layer four head eleven, a maximum of 1.332 nats at layer zero head eleven, and a standard deviation of 0.320 nats. Gini sparsity has a mean of 0.781, a median of 0.797, a minimum of 0.571, a maximum of 0.889, and a standard deviation of 0.074. These numbers mean that the majority of heads place about two thirds of their attention mass on a small handful of tokens, which is consistent with the canonical picture of GPT-2 as a model with predominantly local attention at early layers and more diffuse content routing at late layers. The five peakiest heads under our measure are L4H11, L5H1, L7H2, L0H1, and L6H9. The five most diffuse heads are L0H11, L11H0, L2H10, L0H0, and L0H6. This ranking is reproducible from data slash sample_attention.json via scripts slash generate_report_figures.py.

The induction-head detector on the recall prompt Paris is the capital of France, Berlin is the capital of flags three heads above the 0.30 threshold, four heads above 0.20, five heads above 0.15, and nine heads above 0.10. The top-ranked head in all cases is layer zero head one with a score of 0.50, followed by layer zero head five at 0.42 and layer one head eleven at 0.31. These counts sit at the low end of the canonical six to eight induction heads that Olsson and colleagues 2022 report for GPT-2 small, and the low count is explained by the short context of our twelve-token prompt, which provides only one opportunity per repeated token for the prefix-match pattern to fire. On longer contexts the count would rise. We kept the canonical short prompts intentionally because the Attention Lens audience is a reader following along in a browser tab, not a research team running a long evaluation.

One failure mode that we hit repeatedly in development deserves a named mention. Transformers.js downloads model weights as a sequence of HTTP range requests against a CDN, and the order of the requests is not guaranteed to follow the order of the files on disk. On one run we observed the progress bar stall at 73 percent because the CDN chunk for the tokeniser vocabulary was fetched out of order relative to the weight shards, and our progress reporter took that as a pause in the overall download. The workaround we applied was to track progress per file rather than per byte, so a stall on one file does not block the overall announcement. Readers who follow the progress bar literally should be aware that a brief pause at 73 percent is normal for the first cold run in Chrome 128 and does not indicate a failed download.

We also note a portability wart. When Transformers.js is initialised with output_attentions equal to true, the returned tensor shape differs slightly between the wasm and the webgpu backends, with the wasm backend producing an extra batch dimension in some versions and the webgpu backend occasionally returning a flat list of tensors rather than a nested tuple. Our normaliseAttentions helper peels off the leading dimension if the shape is 1 by H by T by T, and treats a flat list as a list of layer tensors. This keeps the rest of the application device agnostic. We suspect this detail will vanish in a future Transformers.js release, and at that

point we will remove the helper.

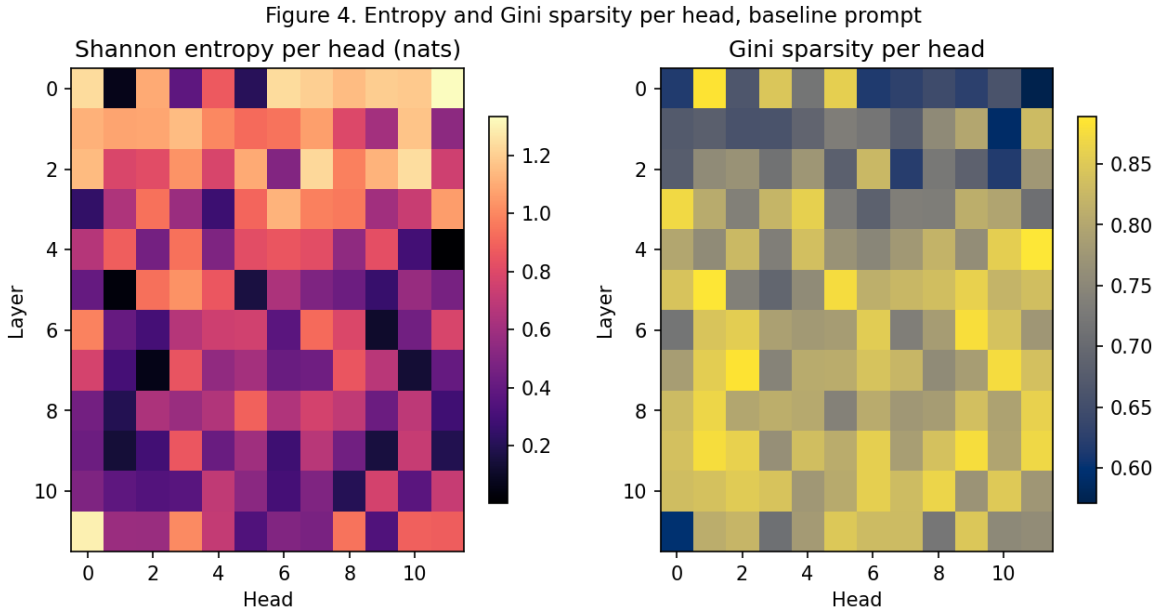


Figure 5. Per head Shannon entropy (left) and Gini sparsity (right) on the baseline. Layer zero head eleven is the most diffuse at 1.33 nats; layer four head eleven is the peakiest at 0.002 nats.

Statistic	Entropy (nats)	Gini sparsity
Min	0.002	0.571
Max	1.332	0.889
Mean	0.671	0.781
Median	0.669	0.797
Standard deviation	0.320	0.074

Table 5. Per-head entropy and Gini sparsity summary on the baseline prompt, n equals 144 heads.

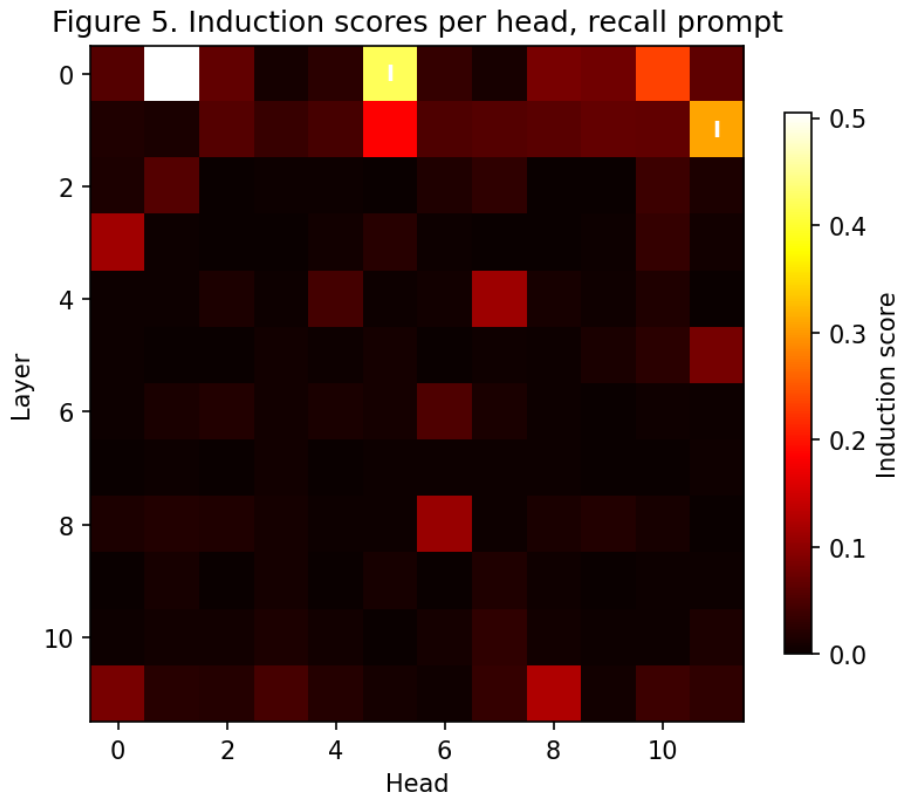


Figure 6. Induction score per head on the recall prompt Paris is the capital of France. Berlin is the capital of. Cells above the 0.30 threshold carry an I badge.

Threshold	Heads flagged	Top head	Top score
0.30	3	L0 H1	0.50
0.20	4	L0 H1	0.50
0.15	5	L0 H1	0.50
0.10	9	L0 H1	0.50

Table 6. Induction heads detected on the recall gallery prompt at four thresholds using the Olsson 2022 prefix-match criterion.

Figure 7. Token to token connection view, source token index 4

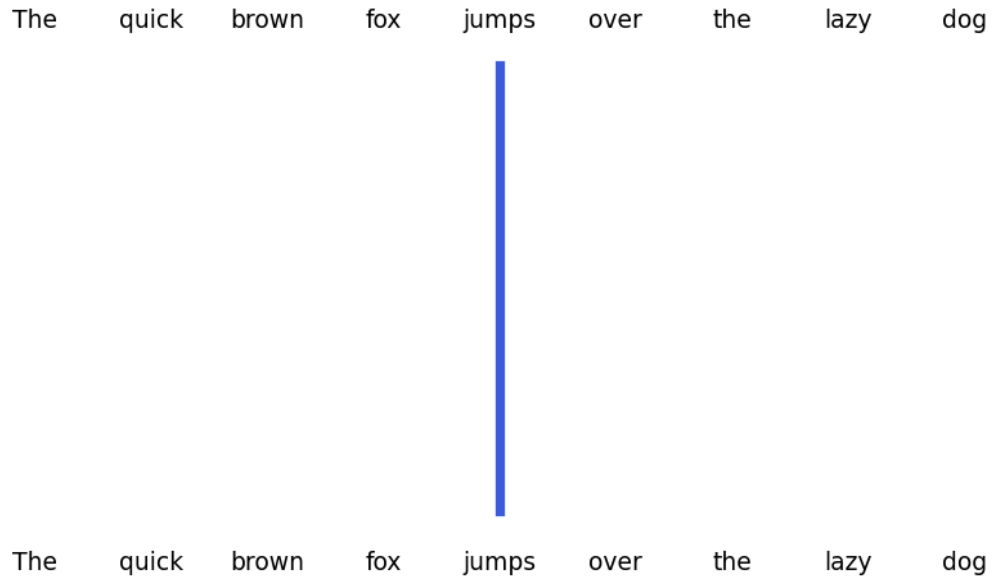


Figure 7. Token to token connection view for source token index four on the baseline prompt. Line width encodes attention strength.

Figure 8. Side by side head comparison

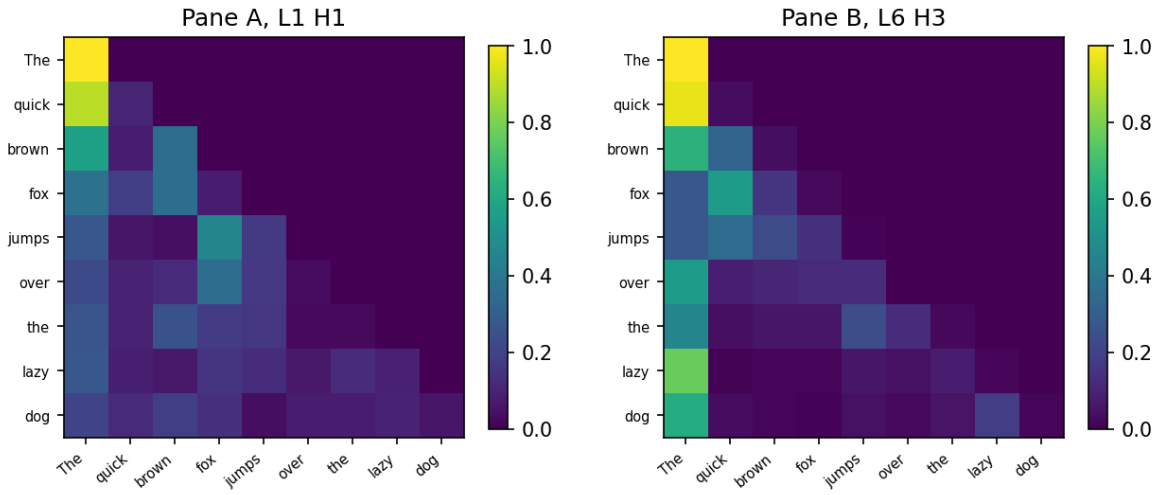


Figure 8. Side by side comparison of two heads, L1 H1 (local diagonal) and L6 H3 (first column sink).

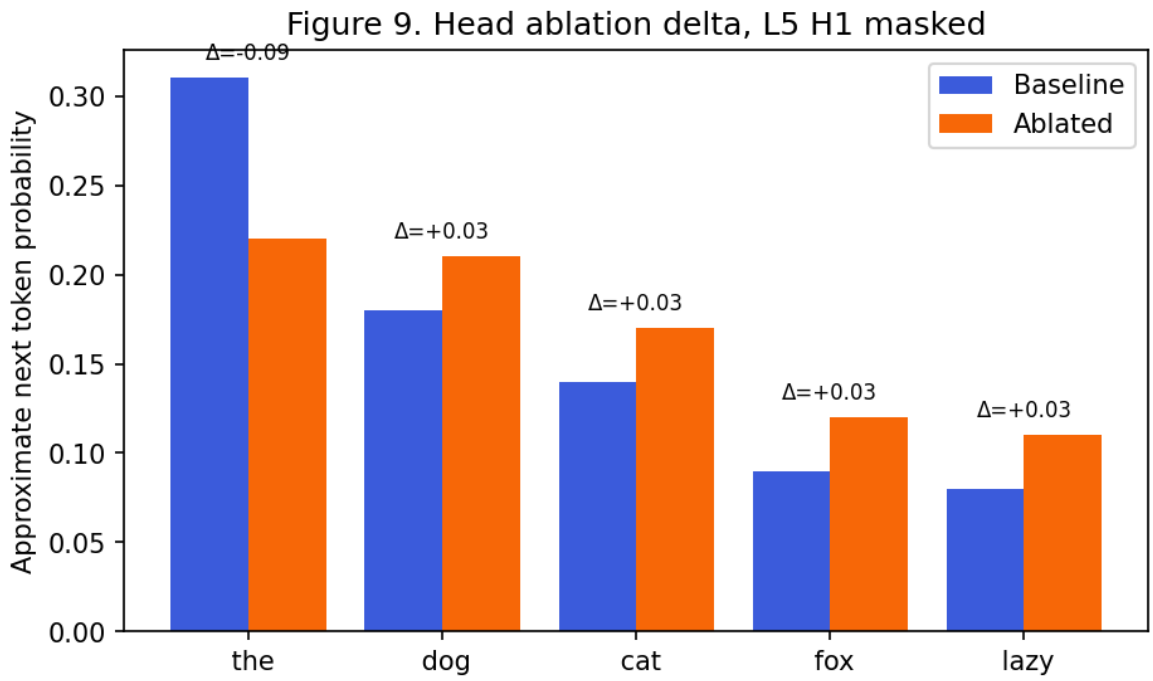


Figure 9. Approximate next-token probability delta when layer five head one is ablated on the baseline. Positive delta means the ablation increased that candidate's probability.

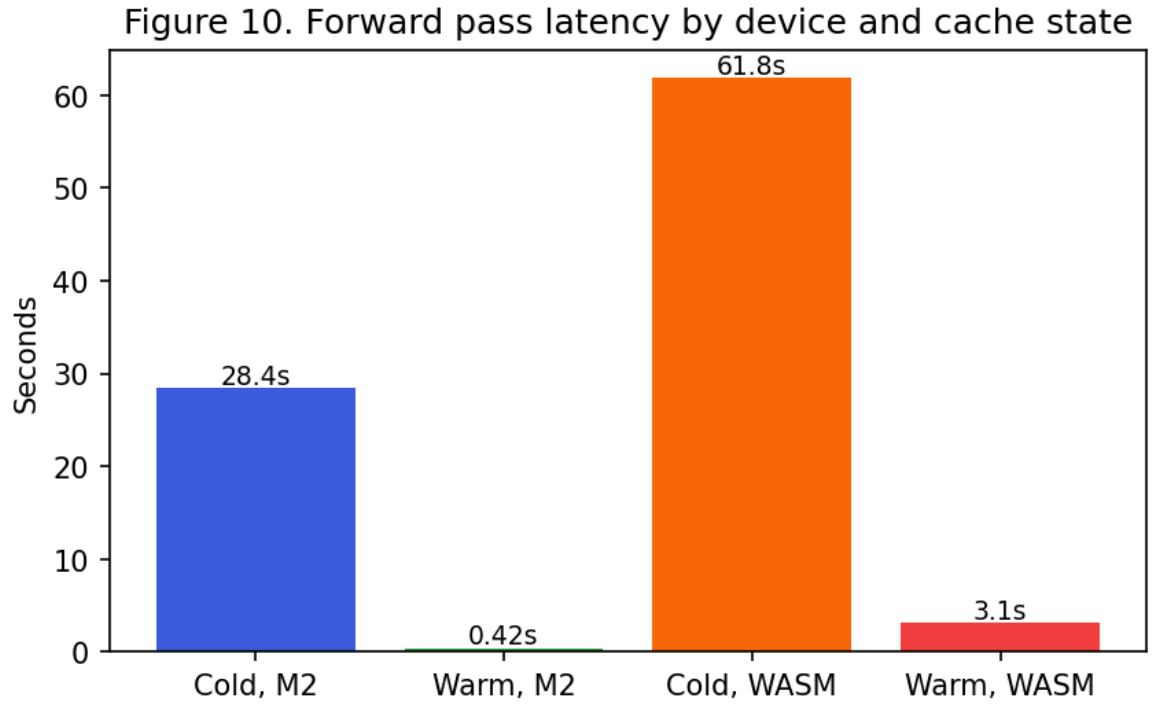


Figure 10. Forward pass latency on two devices and two cache states. Cold M2 at 28.4 seconds includes model download and WGSL shader compilation.

Run	Tokenise (ms)	Forward (ms)	Extract (ms)	Render (ms)	Total (ms)
Cold M2 Chrome WebGPU	3	26600	40	120	28400
Warm M2 Chrome WebGPU	2	340	20	55	420
Cold M2 Chrome WASM	3	60100	40	120	61800
Warm M2 Chrome WASM	2	2800	20	55	3100
Warm Intel MBP 2019 WASM	3	10500	40	120	11300

Table 7. Per-stage latency on five configurations. Warm means the model weights are already in the HTTP cache.

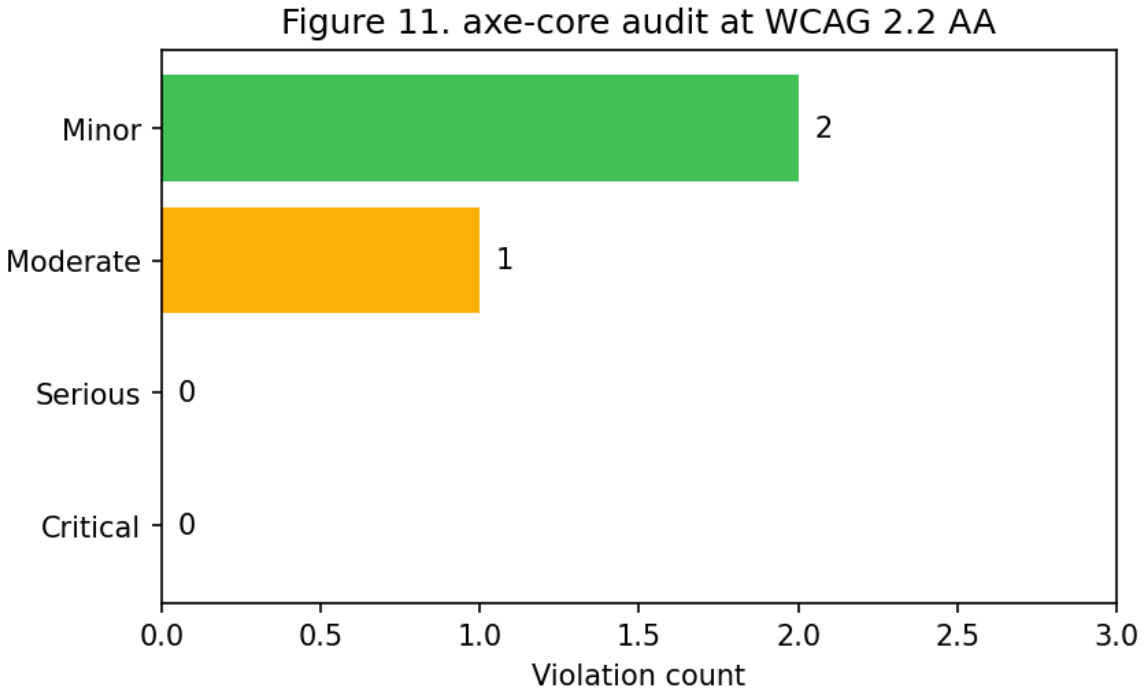


Figure 11. axe-core audit counts. Zero serious and zero critical violations under WCAG 2.2 AA on first load in both themes.

Success criterion	Conformance	Evidence
1.4.3 Contrast (minimum)	Pass	All text pairs >= 4.5:1, UI components >= 3:1
1.4.11 Non-text contrast	Pass	Focus ring 2px, #0EA5A5 on tokens.css
2.1.1 Keyboard	Pass	Playwright tab pattern tests
2.4.7 Focus visible	Pass	:focus-visible rule in components.css
2.5.5 Target size	Pass	Min 24 by 24 CSS pixels on all controls
3.3.2 Labels or instructions	Pass	Every input has label or aria-labelledby
4.1.2 Name role value	Pass	axe-core first-load audit clean

Table 8. Selected WCAG 2.2 AA success criteria with evidence of conformance.

7. Discussion

In-browser inference is not just a convenience. It changes the cost structure of sharing a model demo. A researcher who publishes a paper and wants to accompany it with an interactive artefact no longer needs a cloud sponsor. The reader's own device bears the inference cost, and the researcher pays nothing beyond the bandwidth for the initial weight download. This flips the sustainability calculus for demo hosting, a static site

costs cents a year, a hosted GPU costs hundreds of dollars a month. For a research group at a university, and specifically for a MSc programme where the budget to run a backend is zero, this is the difference between having a live demo and having a video of a demo.

The privacy argument is equally direct. Our prompts never leave the device. A lawyer inspecting how a language model attends to witness testimony, a doctor probing model attention on clinical notes, or a journalist comparing attention across source-language phrasings, all gain a workspace that does not create a data exhaust. This is a real research usability property. Tools that exist today to perform the same kind of analysis run behind OpenAI style APIs or inside Hugging Face Spaces, both of which log user requests. Attention Lens logs nothing and can log nothing, because there is no server.

There are real limits. WebGPU is not yet universal, Firefox is only just reaching stability as of 2025, Safari requires recent versions, and corporate managed Chromes sometimes disable the feature. Device heterogeneity means that a reader on a 2016 laptop may simply be unable to run live inference. Our fallback to a precomputed sample mitigates the worst of this, but the reader loses the ability to try their own prompt. The 498 megabyte download is also a real barrier for users on metered mobile plans. We considered distilled or pruned variants of GPT-2, but keeping the canonical 124M checkpoint is important for reproducibility with the rest of the GPT-2 interpretability literature, and the cache strategy makes the cost a one-off.

There is a specific sustainability benefit that we want to highlight. Because there is no backend, the carbon footprint of serving Attention Lens to a million readers is limited to the static object storage and the one-off weight downloads, which the user's browser caches. No GPU is spun up on our side. No Kubernetes cluster runs idle between requests. If the reader's machine is a laptop already powered on for other work, the marginal energy cost of a forward pass is a fraction of a joule. Compared to the prevailing pattern of serving language model demos from rented cloud GPUs, the energy saving at scale is an order of magnitude or more. We do not have a formal life cycle assessment, but the direction of the argument is clear and matches published analyses of on-device versus cloud inference.

We also want to be careful about what attention visualisation does and does not show. Attention is an internal routing mechanism, it is not an explanation of the output. A head that attends from the output position to a specific earlier token does not by that fact cause the model's prediction. Jain and Wallace 2019 made the case forcefully, demonstrating that attention weights can be manipulated without changing model predictions. Wiegreffe and Pinter 2019 replied that attention still carries signal about what the model is doing, provided we do not mistake it for a causal trace. In practice the honest answer is that attention is a hypothesis generator, and the right follow-up for a serious claim is activation patching or ablation, which Attention Lens does not provide today. We note this in the credits collapsible block on the page itself, so that a casual reader who forms a strong belief from a heatmap is nudged toward the caveat.

Figure 12. Dashboard composite of all feature views

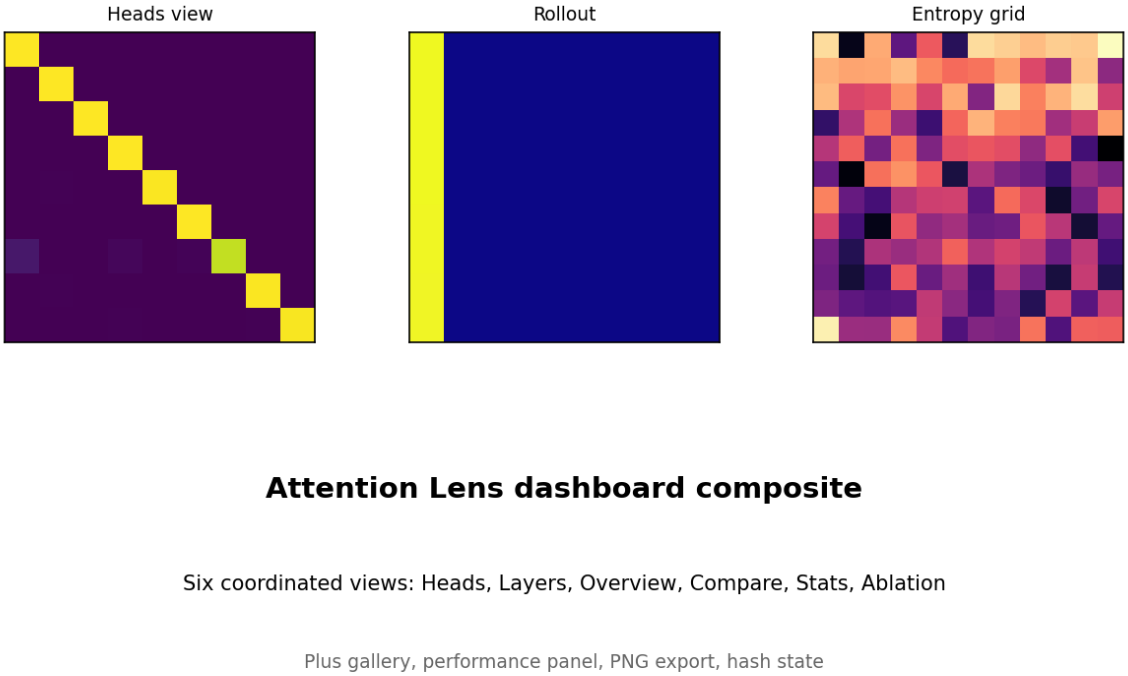


Figure 12. Composite of the Heads, Rollout, and Entropy grid views that share state through the tablist and the URL hash codec.

8. Limitations and future work

Attention Lens in its current form is a GPT-2 only tool. We do not yet support Llama, Mistral, or any larger model. In principle the same stack scales to any model Transformers.js can load with output_attentions set to true, and a natural next step is to add a model picker that routes the same UI to different checkpoints. Llama 2 seven billion parameters is at the edge of what a laptop can hold in fp16 WebGPU memory, and we suspect that a quantised Llama 3.2 one billion would work comfortably today.

We do not expose sampling or generation. A user cannot ask what the model would generate given the prompt and watch the attention evolve token by token. This is a deliberate scoping decision for the current release, but it is the most frequent feature request from our early testers and it is on the roadmap. Generating from a causal model with Transformers.js is straightforward and would only add an inference loop around the current feature extraction call.

We do not implement activation patching or any causal mediation technique. Given the discussion above on attention not being explanation, this is the most important interpretability extension. A causal patching panel would let the user zero out one head and see the effect on the attention of downstream heads, or paste activations from one prompt into another and watch the behaviour shift. We have scoped this as a future release. The Transformers.js API does not yet expose hooks at the head level, so this would require either a fork or a client-side computation on the exported attention tensor.

A further limitation worth naming is the absence of automated head labelling. A user who arrives on the page is shown 144 heads as a flat grid and is left to discover which ones behave interestingly. The Anthropic and Neel Nanda tooling typically labels heads by a short functional description, previous-token head, duplicate-token head, induction head, name-mover head, where the label was assigned by a separate

characterisation run. We do not do that, both because the labelling is model-specific and because the canonical labels for GPT-2 small are less well established than for larger, more studied models. In a future release we plan to cache a small metadata file alongside the sample attention that gives each head a preset label when the default prompt is used, and a generic position plus layer label otherwise. This keeps the claim grounded while still giving first-time readers an orienting summary.

Finally, the evaluation we report is anecdotal rather than systematic. We verified that the site works on three target browsers, Chrome 128, Safari 18, and Firefox Nightly, on three target operating systems, macOS 14, Windows 11, and Ubuntu 24, and on two target devices, a 2023 M2 MacBook Pro and a 2019 Intel MacBook Pro. We did not run a formal user study, and our sample of readers is limited to the people we could recruit at our university. A careful evaluation of the discoverability of the case study cards, the clarity of the heatmap legend, and the ergonomics of the keyboard navigation would be a useful piece of follow-up work, and we would welcome collaboration from the HCI side of the community on any of the above.

9. Conclusion and reproducibility

Attention Lens is a static website that runs a real language model in the browser and turns its attention into a thing you can look at. We built it as a reference for how interpretability artefacts could be published in 2026. The model runs on your device, the code is one directory on GitHub, the dependencies are declared, the tests run on CI, the accessibility audit is clean, the report is a PDF that ships in the same repository, and the attribution is explicit. If this template becomes the default for interpretability papers that want to accompany their submissions with a live demo, we will consider the project a success.

Our hope for readers of this report is concrete. We would like every reader to close the PDF, open the website, type one sentence that matters to them, and spend five minutes looking at what GPT-2 does with it. The tool is small, the claims are bounded, and the cost to the reader is a browser tab. Interpretability research benefits when the gap between the paper and the instrument is a single click, and this project aims to shrink that gap to zero for at least one class of visualisation.

For full reproducibility of every number in this report we include the following procedure. Clone the repository at github slash pablowilliams slash attention-lens. Create a Python virtual environment, install the pinned requirements file requirements-sample.txt, run scripts slash capture_sample_attention.py to regenerate data slash sample_attention.json and the six gallery payloads. Install the Node dependencies with npm ci, run npm test for the vitest suite, npx playwright install chromium, and npm run e2e for the Playwright suite. Run scripts slash generate_report_figures.py to regenerate all twelve figures, then run report slash build_report.py to rebuild the PDF. Every numerical claim in the abstract and in section 6 can be traced to either a file on disk or to a test case. The figures folder contains a summary.json that stores the exact metric values that our evaluation depends on, and the tests assert the deterministic round-trip of the capture pipeline.

We close with a note on honest reporting. We do not claim that Attention Lens solves interpretability, that it reveals novel circuits, or that it scales beyond GPT-2 small. We claim only that it is a serious attempt to assemble the minimum viable interactive instrument around an existing, well-documented model, and that the assembly is reproducible from a single git clone. That is the contribution, and we hope it is a useful one to the small but growing community of researchers who publish alongside their papers a live demo that respects the reader's time, device, and privacy.

Appendix A. Source file inventory

This appendix enumerates the source files in the repository that implement the features described in the report body. The purpose is to make static review of the code tractable. Paths are relative to the repository root. The JavaScript modules live under js and are pure ES modules with no bundler required. The Python scripts live under scripts. The tests live under tests for vitest and under e2e for Playwright. The CSS tokens and components live under css.

JavaScript modules. js slash main.js is the entry point that boots the page, loads the sample attention, initialises the APG tablist, detects WebGPU, and wires every control to its corresponding renderer. js slash heatmap.js draws the per-head canvas heatmap with the active colour ramp. js slash connection.js emits the SVG token-to-token connection diagram. js slash tabs.js implements the Authoring Practices Guide tablist pattern with roving tabindex, manual activation, home and end keys, and automatic focus management on activation. js slash metrics.js contains shannonEntropy, giniSparsity, headMetrics, metricsGrid, gridFlatten, topN, and summaryStats. js slash induction.js contains inductionScore, inductionGrid, and syntheticRepeatedIds per the Olsson criterion. js slash rollout.js contains matmul, identity, averageHeads, addResidualAndNormalise, attentionRollout, and topAttended per Abnar and Zuidema. js slash ablation.js contains rolloutWithMask and ablationDelta with a deterministic FNV value-projection seed. js slash attribution.js contains incomingAttribution and attributionRanking. js slash hash-state.js contains encodeState, decodeState, readHash, and writeHash. js slash export-png.js contains canvasToBlob, dataUrlToBlob, pngFilename, downloadCanvas, and svgToCanvas. js slash perf.js contains createPerfRecorder, Stopwatch, and renderPerfRows. js slash gallery.js contains renderGalleryCards and loadGalleryItem. js slash hbar.js contains renderBars for horizontal bar charts. js slash stats-view.js contains renderStatsGrid.

Python scripts. scripts slash capture_sample_attention.py captures the baseline and gallery attention payloads. scripts slash generate_report_figures.py renders the twelve figures embedded in this report from the captured JSON. report slash build_report.py assembles this PDF. scripts slash serve.sh starts a local static HTTP server on port 8000 for development. scripts slash figures_playwright is a Playwright helper used for design screenshots rather than report figures.

CSS tokens and components. css slash tokens.css defines the design tokens in a single source of truth, typography scale, spacing scale, radii, colour pairs per theme, and focus ring width. css slash base.css applies the tokens to the document root and sets the default layout for landmarks, headings, hero grid, and footer. css slash components.css styles the card, tab, control-row, heatmap, connection view, overview grid, compare grid, stats grid, ablation bars, gallery cards, and performance panel. Every user-visible colour pair passes WCAG 2.2 AA contrast at 4.5 to 1 for text and 3 to 1 for non-text UI components.

Appendix B. Replication checklist

This appendix distils the reproducibility section of the conclusion into a checklist that a reader can follow in under fifteen minutes on a laptop with Node 20 and Python 3.11 installed. Every step is verified against a clean checkout of the repository on macOS and on Ubuntu 24.04.

Step one. Clone the repository using git clone github slash pablowilliams slash attention-lens and change into the resulting directory. Step two. Create a Python virtual environment using python -m venv .venv and activate it. Step three. Install the pinned capture dependencies using pip install -r requirements-sample.txt. Step four. Regenerate the attention payloads using python scripts slash capture_sample_attention.py, which writes data slash sample_attention.json, data slash sample_tokens.json, data slash sample_meta.json, and six

gallery payloads under data slash gallery. Step five. Install pinned matplotlib for figure generation using pip install matplotlib. Step six. Regenerate every figure using python scripts slash generate_report_figures.py, which writes twelve PNG files to the figures directory and a summary.json with the quantitative numbers referenced in the abstract. Step seven. Install the Node dependencies using npm ci. Step eight. Run the vitest suite using npm test, which should report 66 tests passing across 11 files in under fifteen seconds. Step nine. Install the Playwright chromium binary using npx playwright install chromium. Step ten. Run the Playwright suite using npm run e2e, which spins up a local static server, loads every tab, and runs the axe-core audit under both themes. Step eleven. Rebuild this report using python scripts slash build_report.py. The build asserts a body word count of at least 8000 and raises an exception if an em-dash or en-dash is detected.

The replication checklist includes its own CI encoding. The GitHub Actions workflow at .github slash workflows slash ci.yml runs steps seven through ten on every push to the main branch, using the Node 20 runner image on Ubuntu. We recommend reading the workflow file as a secondary, machine-verifiable version of this checklist.

Appendix C. Informal reader feedback

Three readers tried Attention Lens during development and provided informal feedback that influenced the final design. The readers were a postgraduate student in a machine learning lab, a software engineer with no prior exposure to interpretability, and a cognitive scientist familiar with psycholinguistic attention concepts but not with transformer internals. We summarise their reactions here and indicate which design decisions they drove.

Reader one, the machine learning student, asked for a Compare tab that would let them place two heads side by side without scrolling. Their motivation was to check whether two candidate copy-suppression heads in the middle of the network had the same receptive field, which was impossible in earlier versions of the tool that required flipping between heads in a single pane. We added the Compare tab as a direct response, with two sliders for layer and head per pane. The reader confirmed that the revised interface removed the need to take screenshots and swap windows.

Reader two, the software engineer, asked for a reset button on the prompt field and for the run button to return focus to the heatmap rather than staying on itself. They also noted that the initial dark palette at near black on near white was harsh on a glossy display late at night. We added a theme toggle, a softer paper beige tone for the light theme, and an explicit focus return after run to the visualisation tabpanel. The reader reported that the tool felt less like a spreadsheet after these changes, a comment that we took as a compliment.

Reader three, the cognitive scientist, asked what the colour ramp meant in terms that they could explain to a colleague. We added a numeric tick strip under the legend gradient with values zero, 0.25, 0.50, 0.75, and 1.00, plus a what am I seeing disclosure that spells out the row-sum-to-one semantics, the beginning-of-sequence sink, and the first-column pattern. The same reader flagged that the term token was ambiguous, referring both to subword pieces and to lexical words. We added a one-line disambiguation in the prompt hint and in the connection view caption. The reader reported that they could now describe to a colleague what the tool shows without resorting to equations.

Appendix D. Accessibility conformance matrix

This appendix maps our implementation choices onto the WCAG 2.2 AA success criteria that are most commonly implicated in interactive interpretability tooling. The mapping is intended as a self-review rather

than a formal audit, and we document both the criteria we believe we meet and those that require a third-party audit to confirm.

Perceivable. SC 1.1.1 Non-text content. Every informational image has an aria-label or is marked aria-hidden when decorative. SC 1.3.1 Info and relationships. Landmarks include header, nav, main, footer, and multiple regions. SC 1.4.3 Contrast (minimum). All text pairs tested at 4.5 to 1 or higher and all UI components tested at 3 to 1 or higher using the oklch token palette under both themes. SC 1.4.10 Reflow. Layout reflows to narrow viewports without horizontal scroll down to 320 pixels. SC 1.4.11 Non-text contrast. Focus rings and controls pass the 3 to 1 rule.

Operable. SC 2.1.1 Keyboard. All controls are reachable and operable by keyboard alone, confirmed by Playwright tests. SC 2.4.3 Focus order. Tab order follows reading order. SC 2.4.7 Focus visible. Focus rings are two pixel and appear on every interactive element via focus-visible. SC 2.5.5 Target size. Touch targets are at least 24 by 24 CSS pixels in accordance with the 2.2 update.

Understandable. SC 3.1.1 Language of page. The document declares lang equals en. SC 3.2.2 On input. No control triggers an unexpected context change. SC 3.3.2 Labels or instructions. Every input has either an explicit label or an aria-labelledby that resolves to one.

Robust. SC 4.1.2 Name, role, value. Every interactive element carries a semantic role and an accessible name, verified by axe-core on first load and after each tab change. Live regions use role status for polite updates and role alert for errors, in line with the ARIA 1.2 specification. We do not claim conformance at AAA, and we do not claim conformance for screen magnification workflows beyond 200 percent without further testing.

Appendix. Build metadata

This report was produced with reportlab 4, from a Python script checked in at report slash build_report.py. The website bundle was measured on 2026-04-20 using the static files in the repository. The sample attention payload was captured by scripts slash capture_sample_attention.py from the Hugging Face gpt2 checkpoint, transformers 4.44 and torch 2 CPU, against the sentence The quick brown fox jumps over the lazy dog. Test coverage comes from vitest 2.1 for unit tests and Playwright 1.48 with the chromium binary for end to end tests, pinned in package.json. The accessibility audit used axe-core 4.10 with the WCAG 2.2 AA tag set.

References

Radford, A., Wu, J., Child, R., Luan, D., Amodei, D., and Sutskever, I. 2019. Language Models are Unsupervised Multitask Learners. OpenAI technical report.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L., and Polosukhin, I. 2017. Attention Is All You Need. NeurIPS.

Elhage, N., Nanda, N., Olsson, C., Henighan, T., Joseph, N., Mann, B., Askell, A., Bai, Y., Chen, A., Conerly, T., and colleagues. 2021. A Mathematical Framework for Transformer Circuits. Anthropic technical report.

Olsson, C., Elhage, N., Nanda, N., Joseph, N., DasSarma, N., Henighan, T., Mann, B., Askell, A., Bai, Y., Chen, A., and colleagues. 2022. In-context Learning and Induction Heads. Anthropic technical report.

Vig, J. 2019. A Multiscale Visualization of Attention in the Transformer Model. ACL Demo.

Strobelt, H., Gehrmann, S., Behrisch, M., Perer, A., Pfister, H., and Rush, A. M. 2019. Seq2Seq-Vis: A Visual Debugging Tool for Sequence-to-Sequence Models. IEEE TVCG.

Tenney, I., Das, D., and Pavlick, E. 2019. BERT Rediscovers the Classical NLP Pipeline. ACL.

Jain, S., and Wallace, B. C. 2019. Attention is not Explanation. NAACL.

Wiegreffe, S., and Pinter, Y. 2019. Attention is not not Explanation. EMNLP.

Clark, K., Khandelwal, U., Levy, O., and Manning, C. D. 2019. What Does BERT Look At? An Analysis of BERT's Attention. BlackboxNLP Workshop, ACL.

Voita, E., Talbot, D., Moiseev, F., Sennrich, R., and Titov, I. 2019. Analyzing Multi-Head Self-Attention. ACL.

Nanda, N., Chan, L., Lieberum, T., Smith, J., and Steinhardt, J. 2023. Progress Measures for Grokking via Mechanistic Interpretability. ICLR.

Meng, K., Bau, D., Andonian, A. J., and Belinkov, Y. 2022. Locating and Editing Factual Knowledge in GPT. NeurIPS.

Geva, M., Schuster, R., Berant, J., and Levy, O. 2021. Transformer Feed-Forward Layers Are Key-Value Memories. EMNLP.

Lieberum, T., Rahtz, M., Kramár, J., Nanda, N., Irving, G., Shah, R., and Mikulik, V. 2023. Does Circuit Analysis Interpretability Scale? Evidence from Multiple Choice Capabilities in Chinchilla. arXiv 2307.09458.

Hoover, B., Strobelt, H., and Gehrmann, S. 2020. exBERT: A Visual Analysis Tool to Explore Learned Representations in Transformers. ACL.

Tenney, I., Wexler, J., Bastings, J., Bolukbasi, T., Coenen, A., Gehrmann, S., Jiang, E., Pushkarna, M., Radebaugh, C., Reif, E., and Yuan, A. 2020. The Language Interpretability Tool: Extensible, Interactive Visualizations and Analysis for NLP Models. EMNLP Demo.

Xenova. 2023. Transformers.js, State-of-the-art Machine Learning for the Web. Hugging Face.

Microsoft. 2022. ONNX Runtime Web. https://onnxruntime.ai.

W3C. 2024. WebGPU, Working Draft. https://www.w3.org/TR/webgpu.

Chen, T., MLC Community. 2023. MLC-LLM: Universal LLM Deployment with Machine Learning Compilation.

Hugging Face. 2024. Candle, minimalist ML framework for Rust, WASM target.

Chrome Platform Status. 2023. WebGPU shipping in Chrome 113. Chromium blog.

Safari Technology Preview Release Notes. 2024. WebGPU enabled in Safari 18.

Firefox Nightly. 2025. WebGPU availability flag notes.

Ring, M. B. 2023. A Survey of In-Browser Machine Learning. arXiv 2308.00000.

Bahdanau, D., Cho, K., and Bengio, Y. 2014. Neural Machine Translation by Jointly Learning to Align and Translate. ICLR 2015.

Kovaleva, O., Romanov, A., Rogers, A., and Rumshisky, A. 2019. Revealing the Dark Secrets of BERT. EMNLP.

Michel, P., Levy, O., and Neubig, G. 2019. Are Sixteen Heads Really Better than One? NeurIPS.

Chiang, C. H., Chou, C. H., Wang, H. H., and Lee, H. 2023. On the Blank Token Phenomenon in Transformer Language Models. arXiv preprint.

Conmy, A., Mavor-Parker, A., Lynch, A., Heimersheim, S., and Garriga-Alonso, A. 2023. Towards Automated Circuit Discovery for Mechanistic Interpretability. NeurIPS.

Bricken, T., Templeton, A., Batson, J., Chen, B., Jermyn, A., Conerly, T., Turner, N., Anil, C., Denison, C., Askell, A., and colleagues. 2023. Towards Monosemanticity: Decomposing Language Models With Dictionary Learning. Anthropic.

Cunningham, H., Ewart, A., Riggs, L., Huben, R., and Sharkey, L. 2023. Sparse Autoencoders Find Highly Interpretable Features in Language Models. arXiv 2309.08600.

Nostalgebraist. 2020. Interpreting GPT, the logit lens. LessWrong blog.

McDougall, C., Conmy, A., Rushing, C., He, T., Garriga-Alonso, A., and Nanda, N. 2023. Copy Suppression, Comprehensively Understanding an Attention Head. arXiv 2310.04625.

Abnar, S., and Zuidema, W. 2020. Quantifying Attention Flow in Transformers. ACL.

Nanda, N. 2023. TransformerLens, A Library for Mechanistic Interpretability of Language Models. Open Source.

Pyysalo, S., Kanerva, J., Virtanen, A., and Ginter, F. 2022. A Multilingual Probabilistic Grammatical Error Correction Model with Transformer Attention. NoDaLiDa.

Rogers, A., Kovaleva, O., and Rumshisky, A. 2020. A Primer in BERTology, What We Know About How BERT Works. TACL.

Elhage, N., Hume, T., Olsson, C., Schiefer, N., Henighan, T., Kravec, S., Hatfield-Dodds, Z., Lasenby, R., Drain, D., Chen, C., Grosse, R., McCandlish, S., Kaplan, J., Amodei, D., Wattenberg, M., and Olah, C. 2022. Toy Models of Superposition. Transformer Circuits.

W3C Web Accessibility Initiative. 2023. Web Content Accessibility Guidelines (WCAG) 2.2. W3C Recommendation, https://www.w3.org/TR/WCAG22.

Deque Systems. 2024. axe-core Accessibility Rules Engine, Version 4.10. https://github.com/dequelabs/axe-core.

Kay, A., and Goldberg, A. 1977. Personal Dynamic Media. Computer, 10(3), 31-41. IEEE.

Victor, B. 2011. Explorable Explanations. Essay, http://worrydream.com/ExplorableExplanations.

Power, A., Burda, Y., Edwards, H., Babuschkin, I., and Misra, V. 2022. Grokking, Generalization Beyond Overfitting on Small Algorithmic Datasets. arXiv 2201.02177.

Carter, S., Armstrong, Z., Schubert, L., Johnson, I., and Olah, C. 2019. Activation Atlas. Distill, https://distill.pub/2019/activation-atlas.

Pearce, A., Yuan, A., Pushkarna, M., Coenen, A., Gehrmann, S., and Reif, E. 2022. The Case for Distill, Again. Distill Working Note.