ChatGPT

Dev Journal — Multiverse Donut Log

Entry #001

Date/Time (local): 2025-12-10 14:10
Title: “AI, Torus, and the Art of Staying Aligned”
Subtitle: Holographic geometry, field overlays, weak control, XR inputs, and paraconsistent UX – a running chronicle

Summary: - Introduced an **Alignment Mode** toggle (UI badge + hotkey) for an orthographic seam-check view, improving visual stability and confirming the torus’s UV alignment.
- Implemented foundational **field overlays**: added main and solar field shells plus an IIT-inspired **integration band** as new toroidal layers, setting the stage for multi-field holographic visualization.
- Began integrating multimodal **XR input hooks** (gaze, dwell, gesture, voice, EEG) with smoothing and spatial UI anchors, and sketched out **neuroadaptive** feedback cues (edge-of-chaos meter, personalized rhythms, “Aha” insight signals) as future enhancements.
- Outlined group coherence features (optional shared-phase sync feedback and implicate vs. explicate boundary overlays) and updated internal design notes to reflect these paraconsistent, adaptive elements.

Changes/Decisions: - **Alignment Mode & Layout:** Added an Alignment Mode UI badge (with a Ctrl/Cmd+L hotkey) that toggles an orthographic alignment camera view (`data-alignment-mode`) and forces a re-render for seam checking. Styled the badge in the light theme CSS. Also removed the hidden “ghost” sidebar element and enforced a full-viewport canvas, eliminating stray UI gradients and ensuring the 3D view spans the entire screen.
- **Field Overlays:** Implemented new torus *field overlay* shells for the main torus and a secondary “solar” torus, plus an **integration band** overlay (inspired by Integrated Information Theory’s high-Φ concept). Added default parameters in `state.fields` and built these layers during scene initialization. The field overlays will update dynamically as torus dimensions change, laying groundwork for upcoming UI controls (color, opacity).
- **UV Mapping Policy:** Unified UV coordinates across all components – the main torus, SolarOS torus, LoL lattice, and grid overlays – using a single basis with **Repeat** wrapping. This ensures seamless tiling of textures/patterns; Alignment Mode was used to verify that no UV seams are visible across the joined surfaces.
- **XR Input Integration:** Drafted a plan for **XR/input hooks** to enable multi-modal interaction. This includes arbitration logic to fuse inputs from gaze, head-dwell, hand gestures, voice commands, clicks, and EEG signals, with smoothing/hysteresis to prevent jitter. Also envisioned spatial menu widgets anchored to the torus geometry (e.g. floating icons on nodes) and subtle fractal or LoL “halo” cues to indicate modes in an XR setting.
- **Neuroadaptive Hooks:** Added placeholders for **neuroadaptive feedback** mechanisms. For example, an “edge-of-chaos” coherence meter (to signal when the system is at a critical balance), individual alpha frequency alignment (personalized IAF tuning), soft **weak-control** phase nudges (gentle biasing of attention rhythms rather than hard clamps), a cross-band “Aha” pulse (noting when an alpha dip and gamma spike coincide as a simulated insight), and fractal criticality overlays. These features remain

conceptual for now, ready to be activated as the biofeedback loops come online.
- **Group Sync & Boundary Cues:** Envisioned **group synchronization** feedback for multi-user sessions, allowing optional shared-phase indicators if users opt in (e.g. a subtle glow or pattern when two users' attention phases align). Also proposed **boundary overlays** tinting the torus surfaces to differentiate inner vs. outer focus (implicate vs. explicate order), reinforcing the holographic boundary↔bulk metaphor. These conceptual features were documented for future development in our design notes.

Next Steps: - Hook up UI controls to toggle and adjust the new field overlays (e.g. color and opacity sliders for the main field, solar field, and integration band).
- Implement an interface for adding/removing **anchors** on the torus and visualize their positions, tying the anchor system into user interactions.
- Investigate and fix the minor layout glitch (the blank **sidebar** band on the right) to ensure the canvas truly occupies the full viewport.
- Continue developing the XR input and neuroadaptive systems, gradually integrating live EEG/gaze input and criticality metrics once the core visual overlay features are solid.

Entry #002

Date/Time (local): 2025-12-10 14:30
Title: "Alignment Mode lands; fields go holographic"

Summary: - Finalized the **Alignment Mode** feature with a UI badge and hotkey, which toggles an orthographic view for seam alignment verification and forces a clean re-render of the torus.
- Extended the torus visualization with **holographic field overlays**: integrated the main field shell, a solar field shell, and the integration band as additive layers around the donut.
- Cleaned up implementation details (deduplicating the Alignment Mode badge code) and prepared for next enhancements (like field overlay color controls), while flagging a minor layout issue (a blank right-side panel) for resolution.

Changes/Decisions: - **Alignment Mode Implementation:** Enabled the Alignment Mode toggle in the UI (with a badge icon and `Ctrl/Cmd+L` hotkey). Toggling this mode applies an orthographic camera view (setting a `data-alignment-mode` attribute) that makes the torus face the screen, highlighting UV seams, and it triggers a full re-render to ensure a fresh alignment check. The badge was styled in **light.css** for visibility.
- **Field Overlays Added:** Introduced the initial set of field overlay layers. On scene initialization, the system now constructs an outer **main field** shell and a secondary **solar field** shell around the torus, plus the intermediate **integration band**. These overlays are defined in `state.fields` defaults and automatically rebuild whenever the torus's inner radius, tube radius, or related parameters change, ensuring the overlays stay in sync with the geometry.
- **Badge Deduplication:** Removed a duplicate definition of the Alignment Mode badge in the code, streamlining the toggle logic so that only one badge element and event handler exist. This prevents any conflicting behavior or redundant UI elements for the same function.
- **Layout Anomaly Noted:** Observed that an empty "ghost" sidebar on the right was still occupying space and creating a blank vertical strip in the interface. Identified this as a layout bug to address soon (by fully collapsing or disabling that hidden sidebar when not in use).

- **Prepared Enhancements:** Set the stage for further improvements, such as adding UI controls for field overlay properties (color and opacity toggles) and linking the new Alignment Mode and overlays with upcoming anchor and grid features.

Next Steps: - Implement user interface options to customize the field overlays, including toggles to show/hide the field shells and sliders to adjust their color or opacity.
- Fix the blank right-side panel by properly hiding or removing the inactive sidebar element from the DOM/CSS.
- Ensure the Alignment Mode and field overlays interplay smoothly with the planned anchor points and grid overlays (upcoming features for enhanced alignment visualization).

Entry #003

Date/Time (local): 2025-12-10 14:55
Title: "Anchors drop; fields settle"

Summary: - Refined the **field overlay** system: the torus field shells (main, solar, integration band) are now built automatically during scene initialization and update consistently when torus parameters change, solidifying their integration.
- Introduced an **anchor** mechanism on the torus: defined a new state for anchors and created a routine that maps torus coordinates to 3D positions, rendering anchor point markers (sprites) attached to the donut geometry.
- Verified that previous improvements are stable – the Alignment Mode toggle works as intended and the alignment badge deduplication from the last update is confirmed effective.
- Prepared to expose anchors to user control and continued examining the lingering UI layout issue (the blank sidebar panel) for an upcoming fix.

Changes/Decisions: - **Field Overlays Integration:** Adjusted the scene initialization flow so that after `torusSystem.initScene` runs, the code calls `buildFieldOverlays()` and `updateFieldOverlays()`. This ensures that the **main field shell**, **solar field shell**, and **integration band** overlays are present as soon as the scene loads and are sized correctly. Whenever the torus's major or minor radius updates (or other related settings), `updateFieldOverlays` re-calculates these overlay meshes so they remain aligned with the torus.
- **Anchor System Implemented:** Added an **anchor** feature for marking specific points on the torus. Defined a new `state.ui.anchors` configuration with defaults, and built an anchor group generator that takes a set of torus parameters `{u, v}` and converts them to XYZ positions on the torus surface. For each anchor point, an additive sprite (e.g. a small glowing dot or icon) is created and attached to the main torus group (`donutGroup`). This allows visual reference points ("anchors") to be placed on the donut for alignment or annotation purposes.
- **Anchor Utilities:** Created helper functions such as `disposeAnchors` (to remove all current anchors from the scene and clean up their objects) and `rebuildAnchors` (to rebuild the anchor set, e.g., after state changes). Also wrote a utility to translate torus coordinates (like a pair of angular parameters) into actual 3D world positions on the torus, ensuring anchors align correctly to the torus geometry.
- **Confirmation of Fixes:** Confirmed that the Alignment Mode badge deduplication from the previous entry is working – only one alignment toggle is present and it toggles consistently. Also noted that Alignment

Mode itself remains a dedicated seam-check view with no changes in functionality (still activated via the badge or hotkey, and used for verifying UV registration).
- **Upcoming UI Hooks:** Highlighted the need to connect this new anchor system to the user interface. We plan to add controls for enabling/disabling anchors and perhaps a way for the user to drop anchors at will. Additionally, the persistent blank right-side panel (ghost sidebar) issue is still on the radar to fix so that the canvas layout is fully clean.

Next Steps: - Develop UI controls to manage anchors: for example, a toggle to show/hide all anchors and maybe a tool to add or remove an anchor at a selected torus location.
- Resolve the "ghost" sidebar layout issue that is still causing an empty band on the right side of the app, likely by adjusting styles or DOM elements to fully collapse that sidebar when not active.
- Test the interplay between field overlays and anchors (e.g., ensure anchors remain visible above the field shells and do not introduce any memory leaks or performance hits when many anchors are used).

Entry #004

Date/Time (local): 2025-12-10 15:35
Title: "Grids return: LoL + Platonic scaffolds"

Summary: - Reintroduced **grid overlays** to the torus scene, adding three alignment layers: a **Language-of-Light (LoL)** hexagonal lattice wrapped around the torus, a **Platonic scaffold** (using an icosahedron's wireframe as a structural overlay), and an optional **fractal shell** just beyond the torus surface.
- Established a unified `state.grids` data structure with defaults for these overlays and implemented a new `torusSystem.refreshGridLayers()` method to build or rebuild all grid layers whenever the torus geometry or preset configuration changes.
- Ensured proper layering and cleanup: the grid overlays render behind the main torus for a subtle reference, and whenever they are rebuilt, the previous grid meshes and materials are disposed to prevent memory leaks in the Three.js renderer.
- Confirmed that existing features (Alignment Mode, field overlays, anchors) remain stable in combination with the new grids. The known UI issue (a blank sidebar strip) persists, and upcoming work will focus on adding user controls for grid parameters and possibly syncing the LoL grid's motion with system rhythms.

Changes/Decisions: - **LoL Lattice Overlay:** Added a **Language-of-Light** grid overlay around the torus, implemented by creating a torus-knot mesh that forms a continuous hexagonal lattice (LoL) pattern wrapping the donut. This provides a symbolic geometric reference on the torus surface, useful for checking alignment and symmetry.
- **Platonic Scaffold Overlay:** Integrated a **Platonic solid** frame overlay, starting with an **icosahedron**. The edges of an icosahedron are projected around the torus, forming a scaffold that users can use to gauge alignment (the icosahedron was chosen for its symmetry and as a first Platonic layer; more solids can be added later). This overlay is added to a `solidsGroup` in the scene for organizational clarity.
- **Fractal Shell Overlay:** Introduced an optional **fractal shell**, which is essentially a second torus mesh slightly larger than the main torus, with a textured or patterned material that hints at fractal or higher-dimensional structure. This layer is additive and can be toggled independently to provide a visual "aura" or context around the donut.
- **State & Refresh Mechanism:** Created a new `state.grids` object to hold configuration for all these grid

overlays (including booleans for enabling each layer, plus parameters like opacity, scale, rotation, etc.). Implemented `torusSystem.refreshGridLayers()` to manage (re)construction of grid overlays. On app initialization or whenever certain torus parameters change (inner radius, tube radius, or preset shape toggles such as a golden-ratio mode), this function clears existing grid overlays and rebuilds them according to the current `state.grids` settings.
- **Resource Management:** Put safeguards in place so that each time grid overlays are refreshed, any previously attached grid meshes or materials are properly disposed of. This prevents accumulation of hidden Three.js objects (avoiding memory leaks or performance degradation during repeated changes).
- **Render Order:** Adjusted the render order/layering so that the grid overlays appear *behind* the main torus geometry. This way, the grids act as a subtle registration guide and do not occlude the primary torus visualization. The LoL lattice and Platonic edges are visible through or around the torus as faint lines, aiding alignment without overwhelming the view.
- **Pending Issues:** Noted that the right-side blank panel (ghost sidebar) is still visible, indicating the layout fix is yet to be done. Also, while adding the grid overlays, ensured no regressions: Alignment Mode still toggles correctly and field overlays + anchors coexist with the grids without issue.
- **Planned Enhancements:** Identified the need for UI controls to allow users to toggle each grid overlay and adjust their properties (opacity, scale, rotation). Also floated the idea of syncing the LoL lattice overlay with a background cadence or animation (e.g. pulsing or rotating in time with a user's brainwave or a metronome), which remains as an upcoming exploration.

Next Steps: - Develop a **Grid Overlays** control interface (likely as part of the SolarOS panel) where users can enable/disable the entire grid system and each individual layer (LoL, Platonic, fractal) separately, and adjust overlay opacity.
- Add slider controls for fine-tuning each grid overlay's **scale** (to shrink or expand the lattice/scaffold relative to the torus) and **rotation** (to rotate the grid layers around the torus, e.g. aligning a lattice line to a particular anchor or feature).
- Implement a mechanism to **sync the LoL overlay's cadence** with external or internal signals – for instance, tie the grid's rotation or pulsation speed to a specific frequency or to user interaction cues – to enhance neuroadaptive feedback.
- Address the UI layout issue by collapsing or removing the inactive sidebar, so that no blank band appears when the sidebar is not in use (carried over as a priority fix).

Entry #005

Date/Time (local): 2025-12-10 16:05
Title: "Grid toggles land in SolarOS"

Summary: - Implemented a **Grid Overlays** control panel in the SolarOS Hologram UI, giving users the ability to toggle the grid overlay system on/off and individually control each layer (LoL lattice, Platonic scaffold, fractal shell) with visibility switches and opacity sliders.
- Connected these UI controls to the application state (`state.grids`), so any change (toggling a layer or adjusting opacity) immediately calls `refreshGridLayers()` to update the scene and persists the new settings for future sessions.
- Updated the torus renderer logic to honor each overlay's enabled state, ensuring that disabled grid layers are completely hidden and not rendered, thus completing the core integration of grid overlay features into the user experience.

Changes/Decisions: - **SolarOS Grid Panel:** Added a new section in the **SolarOS Hologram** panel dedicated to **Grid Overlays**. This panel provides a master toggle (to enable or disable all grid overlays at once) and individual toggles for each overlay layer: LoL wrap, Platonic scaffold, and fractal shell. Alongside each toggle is an opacity slider allowing the transparency of that layer to be adjusted.
- **State Wiring:** Implemented event listeners for each control that update the corresponding values in `state.grids`. For example, toggling the LoL lattice checkbox sets `state.grids.hex.enabled`, and adjusting its opacity slider updates `state.grids.hex.opacity`. Each of these actions invokes `torusSystem.refreshGridLayers()` behind the scenes, triggering the overlays to rebuild or change material properties immediately. All changes are saved (persisted in local storage under the state key) so that the grid configuration is restored on reload.
- **Renderer Update:** Modified the rendering logic for the torus scene to respect the new state flags. The code now checks whether each grid layer is enabled before attempting to draw it. If a layer is turned off via the UI, the corresponding mesh is not added to the scene (or is removed from it), preventing any hidden or transparent objects from still consuming resources or affecting pick rays. This ensures a true toggle behavior rather than just making things invisible.
- *(Maintained backward compatibility: existing states without grid settings simply default to grids off, and no older functionality was broken by these additions.)*

Next Steps: - Expand the grid overlay controls further by introducing **scale** and **rotation** adjustments for each layer, as planned. This will let users align the LoL lattice or Platonic scaffold at specific orientations or relative sizes.
- Refine the UI layout of the Grid Overlays panel if needed (e.g., grouping controls by overlay, adding labels or icons for clarity). Possibly include preset buttons for common configurations (such as a one-click "Hex Snap" that aligns the LoL grid to Platonic nodes).
- Monitor performance and usability with multiple overlays active. Optimize the `refreshGridLayers()` function if necessary to handle frequent updates (e.g., when sliders are dragged) without frame drops.
- Continue with any remaining UI polish and prepare for integrating more advanced features (like cadence sync or anchor-grid interactions).

Entry #006

Date/Time (local): 2025-12-10 16:18
Title: "Journal now carries code breadcrumbs"

Summary: - Adopted a new documentation practice of embedding **code breadcrumbs** – concise code snippets – into each development journal entry to capture important implementation details and context.
- Provided an initial snippet demonstrating how the grid overlay UI controls in SolarOS are wired into the system (updating state and refreshing the overlays), thereby creating a direct reference for future debugging or refactoring.

Changes/Decisions: - **Code-In-Log Decision:** Decided to include short, relevant code excerpts in the dev journal going forward. By having the actual code that was written for key features right alongside the narrative, we preserve the *intent* and exact approach used, which will help trace issues or recall reasoning in later development stages.
- **First Breadcrumb Example:** Embedded a snippet for the recently added grid toggles. This code shows

how the UI events (toggle switches) update the `state.grids` and call an `applyGridUpdate` helper, which in turn triggers `torusSystem.refreshGridLayers()` and persists the state. This example serves as a template for the kind of breadcrumbs we'll be adding.
- Clarified that these code snippets will remain minimal and focused (not full implementations, but the core logic flow from UI → state → renderer) to avoid clutter. The team will continue this practice for subsequent features to maintain a living technical trail in the journal.

Next Steps: - Continue adding code snippets for new features, especially when implementing the upcoming grid scale/rotation controls and any complex logic (like synchronization or multi-modal input handling).
- Ensure that snippet content stays up-to-date with refactors – if underlying code changes significantly, consider updating or noting it in a future journal entry for accuracy.
- Proceed with developing the grid overlay enhancements (scale and rotation), confident that their key code aspects will be logged for posterity.

Code Breadcrumb:

```
const applyGridUpdate = ({ persistState = false } = {}) => {
  if (torusSystem?.refreshGridLayers) {
    torusSystem.refreshGridLayers();
  }
  if (persistState) persist();
};

gridMainToggle.addEventListener('change', () => {
  state.grids.enabled = gridMainToggle.checked;
  applyGridUpdate({ persistState: true });
});
gridLoLToggle.addEventListener('change', () => {
  state.grids.hex.enabled = gridLoLToggle.checked;
  applyGridUpdate({ persistState: true });
});
```

Entry #007

Date/Time (local): 2025-12-10 16:45
Title: “Grid scales + rotations wired”

Summary: - Expanded the grid overlay controls by adding **scale** and **rotation** adjustments for each overlay layer (LoL lattice, Platonic scaffold, fractal shell), allowing users to fine-tune the size and orientation of these grids relative to the torus.
- Updated the application state and rendering logic to support these parameters: each grid layer now stores a scale factor and rotation angle (UI inputs in degrees are converted to radians for Three.js), and the torus refresh function applies these transforms when rebuilding the overlays.
- Ensured that the new controls work in real-time without page reloads, giving immediate visual feedback.

This enhancement makes alignment checks and overlay layering far more interactive and sets the stage for syncing the LoL lattice with dynamic cues in the future.

Changes/Decisions: - **UI Controls for Scale/Rotation:** Introduced new UI sliders/inputs for **grid scale** and **grid rotation** under each overlay's controls. For example, the LoL lattice overlay now has a rotation slider (0–360°) and a scale slider to expand or contract the lattice. Similarly, the Platonic scaffold and fractal shell overlays received their own scale and rotation controls.
- **State Extensions:** Augmented the `state.grids` structure to include `scale` and `rotation` properties for each layer. Defaults were provided (e.g., scale = 1, rotation = 0) to maintain the previous appearance if a user doesn't tweak these. The rotation is stored in radians internally, although the UI presents it in degrees for usability.
- **Event Binding:** Implemented helper functions (e.g., a generic `bindGridRotationDeg`) to handle binding the rotation slider inputs. These functions convert the input degree value to radians (`THREE.MathUtils.degToRad`) and update the `state.grids[<layer>].rotation`. Likewise for scale, inputs are clamped and directly set `state.grids[<layer>].scale`. Each update calls our existing `applyGridUpdate` (which triggers `refreshGridLayers()` and optionally persists the state) so changes take effect immediately and are saved.
- **Renderer Updates:** Modified the grid overlay construction logic to utilize the new scale and rotation from state. For instance, when creating the LoL lattice geometry (torus knot), we now multiply the base radius by the LoL scale factor, and after creating the mesh, we rotate it by the specified rotation angle (`hexMesh.rotation.y = state.grids.hex.rotation`). Similarly, the Platonic scaffold group is rotated, and the fractal shell torus geometry is scaled/rotated based on its state values. This ensures the 3D scene reflects the UI settings accurately.
- **Interactive Alignment:** Verified that adjusting scales and rotations yields the expected visual results. Users can now, for example, rotate the LoL lattice so that a lattice line aligns with a particular anchor or feature on the torus, or scale down the Platonic scaffold to better fit a certain torus size. These controls greatly enhance the ability to achieve precise **alignment configurations** without code changes.
- **Future Sync Note:** With the manual controls for rotation in place, we noted that a future step will allow an automatic driving of these rotations (especially the LoL overlay) by an external cadence or signal (for example, linking it to a brainwave frequency or a timed cycle). This remains an open experiment to tackle once basic features are complete.

Next Steps: - Implement a feature to **synchronize the LoL grid overlay's rotation or pulse** with live data or predefined rhythms (e.g., tie it to EEG alpha oscillations or a breathing pace), to explore neurofeedback possibilities.
- Polish the UI for the new scale and rotation controls: add numeric readouts next to sliders, allow double-click to reset to default (scale=1, rotation=0), and ensure the control layout remains user-friendly as complexity grows.
- Perform extensive testing with various combinations of grid settings (all overlays on, rotated at unusual angles, etc.) to ensure there are no rendering artifacts or performance bottlenecks. Optimize the refresh logic if needed for smooth adjustments.
- Integrate any user feedback on the overlay controls into iterative refinements, and start considering other overlay types or anchor-to-grid "snap" functionalities as hinted in the design docs.

Code Breadcrumb:

```
const bindGridRotationDeg = (inputEl, numberEl, path) => {
  const toRad = (deg) => THREE.MathUtils.degToRad(Number(deg) || 0);
  const handle = (deg, { persistState = false } = {}) => {
    const target = state.grids[path] || (state.grids[path] = {});
    target.rotation = toRad(clampGridNum(deg, -180, 180, 0));
    applyGridUpdate({ persistState });
  };
  // ...
};
bindGridRotationDeg(gridLoLRotation, gridLoLRotationValue, 'hex');

// Renderer side
const hexGeo = new THREE.TorusKnotGeometry(mainRadius * hexScale, tubeRadius *
0.05, 180, 36, 3, 2);
hexMesh.rotation.y = hexRot;  // apply state.grids.hex.rotation (in radians)

const fracGeo = new THREE.TorusGeometry(mainRadius * 1.02 * fracScale,
tubeRadius * 0.12 * fracScale, 24, 160);
fracMesh.rotation.y = fracRotation;
```

Entry #008

Date/Time (local): 2025-12-10 17:05
Title: “Ghost sidebar collapse to fix blank band”

Summary: - Resolved the persistent UI layout issue by fully collapsing the hidden right-hand “ghost” sidebar, thereby removing the unintended blank gradient band that was appearing on the side of the canvas.
- Applied a CSS tweak to hide the inactive sidebar element completely (setting it to `display: none` when not in use), ensuring the main canvas now extends edge-to-edge without any overlay artifacts.

Changes/Decisions: - **Ghost Sidebar Fix:** Updated the CSS rules for the inactive sidebar (the element we refer to as the “ghost” sidebar, which is present but `aria-hidden="true"` when no panel is open). The new CSS explicitly hides this element by setting `display: none !important` in addition to `visibility: hidden` and disabling pointer events. This removal from layout flow collapses the empty space that was previously reserved for the sidebar’s translucent gradient.
- **Visual Outcome:** With the ghost sidebar effectively removed when not needed, the dark vertical band on the right side of the screen has disappeared. The torus visualization can now use the full browser width, and the overall UI looks cleaner and more immersive.
- **No Side-Effects:** Confirmed that this change does not negatively impact functionality – when the sidebar is supposed to be shown (for example, if a user opens a settings panel that utilizes that area), it will still appear as intended. The fix only targets the scenario when the sidebar is inactive. This improvement addresses a small but noticeable UX issue, polishing the interface.

Next Steps: - Double-check all UI layouts in various modes (with side panels open/closed, in XR fullscreen mode, etc.) to ensure that no other hidden elements cause layout shifts or unused space.
- Return focus to feature development now that the layout distraction is solved: for instance, proceed with experiments on LoL overlay cadence synchronization or start tying the anchor system into the overlay grids (e.g., snapping anchors to the nearest lattice node if applicable).
- Continue refining the **neuroadaptive feedback** elements and begin planning integration with live input devices (EEG headsets, etc.), making sure the visual overlays (fields, grids, anchors) can reflect real-time cognitive metrics in a meaningful way.
- Maintain the updated journal practice by documenting any complex changes with code snippets and ensuring all design changes are mirrored in the project's documentation for consistency.

Code Breadcrumb:

```
.sidebar--ghost[aria-hidden="true"] {
  visibility: hidden;
  pointer-events: none;
  display: none !important;  /* collapse hidden ghost sidebar */
}
```
