ChatGPT

Holographic Tensor Networks and AdS/CFT Tilings

MERA and Hyperbolic Tilings: *Multi-scale Entanglement Renormalization Ansatz* (MERA) networks naturally form a hierarchical, tree-like topology. This has a striking visual analogy to hyperbolic tessellations of a disk [1]. In a MERA, data is processed in layers of increasing scale: moving deeper into the network (toward the "bulk") corresponds to coarse-graining entanglement, while the outermost layer represents fine-grained "boundary" data [1]. Physicists have noted that MERA's layered graph resembles a discrete Anti–de Sitter/Conformal Field Theory setup, where the boundary encodes a bulk via a holographic mapping [2]. In fact, **Swingle (2012)** conjectured that MERA encodes a *discrete* AdS/CFT correspondence, with the MERA lattice playing the role of a curved AdS spacetime and the outer nodes representing a CFT on the boundary. This insight paved the way for explicit *holographic tensor networks*.

HaPPY Code – A Holographic *Pentagon* Tiling: A concrete example is the *Pastawski–Yoshida–Harlow–Preskill (HaPPY) code*, which uses a *hyperbolic tiling* of the disk to encode a quantum state holographically [3]. The construction uniformly tiles an AdS-like disk with **perfect tensors** (each a small quantum code) arranged in pentagons and hexagons [3]. One leg of each tensor is left dangling at the outer boundary, representing a physical qubit; the innermost tensor (a central pentagon) represents the single logical qubit in the "bulk." In essence, a minimal central tensor (the core logical bit) fans out through layers of entanglers to produce a highly redundant set of boundary bits [3]. This network is *holographic*: any given bulk operator's information is spread across many boundary degrees of freedom. Notably, the HaPPY code is an *exact* holographic error-correcting code – even if a portion of the boundary is erased, the logical bulk state can be recovered from the remainder. The tiling used in HaPPY (a tessellation by pentagons and hexagons) is reminiscent of a Penrose pattern on the hyperbolic plane. Each perfect tensor (pentagon) connects to five neighbors, and four pentagons meet around each vertex – a pattern only possible in negatively curved geometry. This model elegantly demonstrates **boundary↔bulk mapping**: a single bulk qubit's information is encoded as a highly entangled patch of many boundary qubits [3]. It draws from AdS/CFT in that the bulk "code space" is protected by the redundancy on the boundary, similar to how quantum error-correcting codes might underlie spacetime geometry.

Penrose Tiling and Holographic Codes: Recent research pushes this idea further by using *quasi-crystalline Euclidean tilings* (like Penrose tilings) as tensor networks. In late 2023, **Li and Boyle** showed that a standard Penrose tiling of the 2D plane can itself be viewed as a quantum error-correcting code [4]. In their construction, quantum information is encoded "through quantum geometry" of the Penrose pattern. The code is remarkably robust: *any* local error or erasure, in *any finite region* of the tiling, can be detected and corrected by the code's structure [5]. In other words, the Penrose tiling inherently provides a form of holographic redundancy – every piece of the pattern contains information about the whole. This result underscores how a minimal seed of information, distributed in a non-periodic, self-similar way, can yield extreme redundancy on the "boundary." The Penrose tiling's aperiodicity is key to this: it is *multi-scale* and never exactly repeats, yet it has hierarchical local patterns suggestive of self-similarity [6]. Each deflation (rescaling) of a Penrose pattern reveals the same motifs at larger scales, much as a MERA adds layers. This quasi-fractal structure means a small set of rules (the Penrose matching rules) generates a richly structured infinite pattern. The *core mechanism* here is an inflation rule that governs how tiles force each other's orientations – effectively a simple code that produces complex global order. The **boundary expression** of such a code is the quasi-crystalline pattern itself, which can cover an arbitrary 2D field without periodicity. Because of the high redundancy, a logical datum could be embedded in the center and recovered from the boundary tiles even if some tiles are removed, akin to a hologram. This *Penrose code* can be seen as a Euclidean analog of the HaPPY hyperbolic code, trading constant negative curvature for an aperiodic order that still expands combinatorially towards the "boundary." The authors also construct variants of the code based on other quasicrystals (Ammann–Beenker tiling with 8-fold symmetry, Fibonacci chain tilings, etc.), including versions that **live on finite periodic boundaries like a torus** [5]. This is a notable bridge between hyperbolic-like redundancy and *periodic boundary conditions*, showing that even on a torus (which has no edges) one can have a holographic-like code by using a quasicrystal pattern wrapped onto it.

Core Mechanism: *Holographic tensor networks* (MERA, HaPPY, Penrose code) all rely on a **minimal core tensor or rule** that expands outwards. In MERA/HaPPY, the core is a central tensor or logical qubit whose influence propagates through layers of unitary gates. In Penrose/quasicrystal codes, the "rule" is the tile substitution or matching rule that propagates local constraints outward to tile the entire space. In each case, simple local transformations (entanglers or matching rules) iterate recursively to fill a large space with structured data. Crucially, no central controller is needed beyond the seed: complexity *emerges from the connectivity*. This resonates with the *Donut of Attention* concept: a small latent focus (central seed) could project outward into a complex pattern of attention weights or memory slots around the ring.

Boundary Expression: The *boundary* in these networks is typically a highly redundant outer layer or surface encoding. In AdS/MERA models, the boundary is a high-resolution description (many degrees of freedom) that all encode the fewer degrees of freedom at the center. In HaPPY's pentagon code, for example, one logical qubit's state is expressed in hundreds of physical qubits on the boundary [3]. In the Penrose tiling code, the "boundary" could be thought of as an ever-expanding region of tiles, or practically, the outer perimeter of a large finite patch – which contains encoded information that originated from the center. This center→boundary mapping is **fractal-holographic**: any given boundary region reflects the influence of deeper structures. The boundary patterns are often visualized as tessellations (circular in hyperbolic space, or irregular in quasicrystals) that are intuitively holographic: zooming into a portion can reveal pieces of the entire code's message.

Torus Mapping Potential: Traditional AdS/CFT tensor networks live on open disks (with a physical boundary). However, for a *toroidal* interface (Donut), we consider closed surfaces with no edge. One approach to put a holographic network on a torus is to use **periodic boundary conditions**, effectively gluing the edges of a finite patch. This generally breaks the perfect self-similarity – for example, a Penrose tiling cannot tile a torus *exactly* without mismatch because it's aperiodic. But as noted above, variants like the Ammann–Beenker tiling can be adapted to a finite torus surface [7] (essentially by choosing a patch that "wraps" consistently). In hyperbolic codes, one can also move to higher-genus surfaces: a *double torus* (genus 2) or beyond can support a true hyperbolic metric, allowing a tiling with negative curvature on a closed surface (this is the idea behind *closed hyperbolic surface codes* in quantum error correction). In fact, families of hyperbolic surface codes on higher-genus surfaces significantly outperform simple toric (genus-1) codes in error correction efficiency [8]. For a single torus (genus 1), a pragmatic approach is to use a **flat tiling with wraparound**. For example, one could take a large square network (which might approximate part of a hyperbolic tiling or a quasicrystal) and identify the opposite edges. This yields a toroidal network, though it introduces two cycles (one around each periodic direction) that might not perfectly align with the intended center→edge logic. Nonetheless, the *DonutOS* could treat one point on the torus as a notional "center" of attention, with patterns radiating outward until they meet and wrap around. The periodic nature means that "boundary" information re-enters from the other side – conceptually, the attention pattern could wrap around and overlap with itself. Maintaining intuitive center→boundary logic on a torus might involve smoothly tapering the influence of a center seed so that it naturally repeats without a hard edge (imagine a radial gradient that eventually repeats at the 360° mark). Known proposals in visualization include *mapping a Poincaré disk to a torus by identification*. This often results in a distorted torus (since a true mathematical mapping would require curvature change), but for an interface, one could *warp* a hyperbolic tiling onto a ring-shaped surface for aesthetic effect. In short, while a torus does not have a distinguished boundary, we can designate an arbitrary loop as the "outer perimeter" for visual purposes – or use one circular direction as radial. The other circular direction would then correspond to wrapping around at a given scale. There have been experimental visualizations of hyperbolic tilings on immersive surfaces; for instance, one can take a {4,6} hyperbolic tiling (squares, 6 around each vertex) and project it onto a genus-2 surface in 3D [9]. For a genus-1 torus, using quasicrystal patterns or repeating lattices is more straightforward. A **practical UI attempt** could be to tile the donut with a Penrose pattern (finite patch) and accept a seam where it wraps, or to use a repetitive but fractal-like pattern (like a *periodic circle pattern* that mimics a fractal at increasing radii). The key is preserving a sense of a *center in the middle of the donut's tube* that influences the "membrane" around it.

UI Use: These holographic network ideas are directly relevant to the **Donut of Attention** interface. They suggest that a toroidal "membrane" can display a *tiling of attention*, where the central focus (user's main attention or a core concept) is encoded across many peripheral tiles (sub-topics, related tokens, etc.). For instance, one could imagine the donut's surface covered in a Penrose-like pattern of nodes, each glowing with intensity reflecting some attention weight. The central idea's influence might propagate through a MERA-like structure: nearest the core are coarse features (broad contexts), which then branch into finer details toward the outer rings. In a co-pilot scenario, this could mean the core command or query is at the donut's inner radius, and the outer surface shows a highly detailed distribution of related code snippets or data points, redundantly encoding the core. Because the code is *error-correcting*, the UI would be robust to occlusion or gaps – e.g. if part of the donut UI is occluded by another window (analogous to erasing some boundary nodes), the central info can still be inferred from the visible part (the remaining pattern). This is an intriguing notion for an attention-steering UI: it would be *forgiving* (information isn't localized in one spot, but distributed), and it reinforces the idea that **every piece of the UI reflects the whole** (a fractal hologram). The user could interact with any tile on the donut's surface and, thanks to the holographic code, be essentially manipulating the global state. This yields a *cognitive mirror*: the donut acts like a mindmap where touching one region can recall the entire focus, since the pattern is constructed with global redundancy. Implementing such a network in DonutOS could involve an underlying graph of attention heads connected in a Penrose/MERA topology, with real-time visualization of activation. The fractal tiling motif would help users intuitively navigate scale: zooming into a portion of the donut might reveal another layer of detail (just as MERA would add layers when you zoom in).

Tensor Networks on Platonic Graphs and Quasicrystals for Fields

Beyond AdS tilings, researchers have explored encoding fields on **Platonic solids and quasi-crystalline grids**. The idea is to discretize 2D or 3D continuous fields (images, volumes, world states) using highly symmetric or aperiodic graphs, where a small set of parameters generates a large structured output.

- **Platonic Graph Codes:** A Platonic solid (tetrahedron, cube, octahedron, dodecahedron, icosahedron) offers a symmetric graph on a closed surface. For example, the dodecahedron (12 pentagonal faces) can host a tessellation akin to a tiny spherical tiling. In quantum error correction, there is indeed a *dodecahedral code* (related to the homological CSS codes family) – sometimes whimsically called a "small stellated dodecahedron code" – which shows how logical qubits can be encoded in the topology of a polyhedral graph [10]. While these Platonic codes are small (because a Platonic solid has a fixed number of vertices), one can imagine a *refinement* process: e.g., subdivide each face of an icosahedron to create a high-resolution sphere (much like geodesic dome construction). The resulting graph inherits icosahedral symmetry and can encode a field on the sphere. This is relevant for world modeling: many planet-scale simulations (weather, global illumination) use geodesic grids or Goldberg polyhedra to sample a field uniformly on a sphere. The **core mechanism** is symmetry – a minimal seed geometry (a Platonic solid) is recursively subdivided to produce a highly regular mesh. This ensures uniform coverage and often yields elegant mathematical properties (equal adjacency, uniform vertex degree, etc.).

- **Quasi-Crystalline 2D Grids:** We discussed Penrose tilings as one example. In general, quasi-crystals (aperiodic tilings) in 2D and 3D are fascinating for encoding because they exhibit long-range order *without repetition*. A quasi-crystal pattern can fill space densely, and often has self-similar subsets. For instance, any finite patch of a Penrose tiling appears infinitely many times in the tiling, albeit in different contexts – this property is useful for error correction and encoding redundancy. Another example is the **Ammann-Beenker tiling** (with 8-fold symmetry) or the **Fibonacci chain** (1D quasi-periodic sequence); Li and Boyle's work suggests these can be used to build codes that fit on a torus or higher-dimensional spaces [7]. The *boundary expression* of quasi-crystal encodings is a dense, non-repeating field pattern that can adapt to arbitrary sizes (since there's no unit cell periodicity). This means you can have a large interface that looks richly textured with no obvious tiling seams – aesthetically pleasing and symbolically appropriate (it looks complex, like the mind's activity, yet it's generated from simple rules).

- **3D Fields and Holographic Lattices:** For 3D volumes, one can use tensor networks like *3D MERA* or spin networks. A *Platonic 3D lattice* example would be a cubic grid (periodic in x,y,z), but more exotically, one might use a quasi-periodic stack of Penrose planes to fill 3D space or even a 3D quasi-crystal (like projections of higher-dimensional lattices – e.g., the E8 lattice projection yields quasicrystalline 4D→3D structures which some physics theories toy with). Models where a **minimal central element governs a huge volume** are seen in machine learning too (discussed below): e.g., a small latent vector can generate a full image via a network. In physics, one could think of a single "core" spin influencing an entire spin network through iterative interactions.

Core Mechanism: In these Platonic and quasi-crystalline encodings, the mechanism is typically **iterative substitution or symmetry operations**. For Platonic graphs, applying a refinement rule (e.g., split each polygonal face into smaller ones) repeatedly creates a finely detailed mesh from an initial polyhedron. The initial polyhedron's symmetries ensure the refined mesh has structured redundancy (e.g., many vertices share similar local configurations due to symmetry). In quasi-crystals, a small set of *prototiles* with matching rules can generate an infinite tiling through inflation/deflation. This is a minimal rule-set (perhaps just two shapes and an inflation substitution rule) that yields a complex field. The **minimal code** could be seen as the prototile set or the generator matrix of the lattice. In either case, the complexity of the field is not explicitly stored tile-by-tile; rather, it emerges from applying the rule across space. This is analogous to a fractal algorithm or even a short piece of code that, when executed, produces a huge dataset.

Boundary Expression: The result of these rules is a **field of values (2D or 3D) that is filled in a structured way**. For example, a Penrose tiling encoding might assign a vector or a scalar to each tile such that the entire tiling represents some function over the plane. Because of the tiling's properties, large-scale features of the field are influenced by the core rules. If we treat the outer surface of a refined Platonic graph as a "boundary," it's a closed, redundant representation of some interior state. Notably, such boundaries can be *highly redundant* – the Penrose code example showed that information is smeared so much that any local cutout can be fixed from the rest [4]. In a UI, this means the displayed pattern could carry information in a diffuse way: you'd read it not from one exact position but from holistic visual cues. (For instance, an interference pattern or a kaleidoscopic image might encode data such that no single piece is fully meaningful until you see the whole.)

Torus Mapping Potential: Platonic solids map naturally to closed surfaces (spheres or higher-genus if we use multiple holes through connected sums). A single torus is topologically equivalent to a plane with periodic boundary—so any **Euclidean tiling** can map to it if it repeats periodically. For quasi-crystals, true periodic repetition is impossible; however, one can choose a finite patch and wrap it. If the patch is large and the join is handled cleverly (perhaps by ensuring the pattern on one edge "almost" matches the other), the discontinuity might be minor or stylized. As mentioned, certain quasi-crystals have been adapted to finite periodic grids [7], effectively finding a large fundamental domain that tiles a torus. On the other hand, Platonic refinements (like geodesic domes) inherently produce closed surfaces. An interesting approach could be to use a **tessellation of the torus by identical shapes** – for example, a donut can be tiled by hexagons if we allow some distortion, or by squares in a grid-like manner. One known result is that a *square torus* can host any periodic tiling that a plane can; so we could use a highly symmetric wallpaper pattern on the donut. If we desire a fractal touch, we might use a *scale-symmetric pattern* that repeats at multiple scales. For instance, imagine a torus textured with a pattern that looks like a Penrose tiling at large scale, but if you zoom in, each "tile" is filled with a smaller copy of the entire pattern (such recursive textures exist in art). This would create a visual hologram effect on the donut's surface. Periodic boundary conditions mean that the notion of "center vs boundary" becomes relative – one might designate an arbitrary point on the torus as the "core focus" and measure distance radially from it (wrapping around if needed). There have been proposals to visualize hierarchical data on a torus by using one angular direction for one hierarchy level and the other angular direction for another (imagine mapping a 2D tree into a torus such that going around the donut one way traverses branches, and the other way cycles through depth). While less common than on a plane, such mappings are conceivable.

UI Use: In DonutOS, these patterns could manifest as **membrane overlays** – backgrounds or grids on the donut that are not merely decorative but informative. For example, a *quasi-crystalline grid overlay* might be used for positioning information widgets. Because quasicrystal grids have no translational symmetry, there won't be obvious straight lines cutting across the donut; this can avoid visual aliasing and give a sense of organic, brain-like structure. Imagine the donut's surface covered in a Penrose tiling where each tile is a UI element (a mini-panel or an icon) – they'd be grouped in quasi-regular pentagons and stars, suggesting a hierarchy without rigid rows or columns. A *Platonic graph* approach might involve something like an "icosahedral geodesic overlay" on a spherical section of the UI (if DonutOS uses any sphere or globe metaphor in addition to the torus). In any case, **minimal latent codes generating complex fields** inspire the system's *world modeling*: the OS could store a small parametric description (like a seed and rule set) and dynamically generate huge attention maps or memory layouts from it. This is akin to how **procedural generation in games** works (tiny seeds create entire worlds), and how **fractal image compression** works (a small equation describes a detailed image). For the co-pilot design pipeline, one could incorporate generative rules so that, for instance, specifying a "5-fold symmetric attention layout" could automatically produce a Penrose-like control panel – the designer only tweaks the seed and rule, and the system fills in the details. Such approaches would make the UI *scalable*: it could zoom in infinitely to reveal more structure, just as fractals do. They also resonate with a *holographic philosophy*: instead of hard-coding every button or link, you define an underlying network and let the interface emerge, maintaining coherence across scales.

Visualizing Networks on a Torus (Donut) Surface

A key challenge for the Donut of Attention is how to **wrap these networks onto a toroidal surface** without losing the intuitive center→boundary metaphor. The torus has no edges – it's a continuous loop in two directions – so if we are to have a "center," we must impose one (much like choosing a pole on a globe). Here we explore approaches and known attempts for toroidal mappings:

- **Periodic Lattice on a Donut:** The simplest tiling of a torus is by identifying opposite edges of a rectangular grid. For example, an $N \times N$ square grid, when opposite edges are glued, becomes an $N \times N$ torus tiling. Many regular patterns (checkerboards, hexagonal grids, etc.) can thus tile a torus. If we use a *flat* tensor network like a 2D grid or a cylinder, we can connect the ends to form a donut. This yields a homogeneous, repeating structure – great for symmetry, but it might lack a focal point. One idea is to use a **moiré or interference pattern** on the torus: by overlaying two periodic patterns slightly offset, we get a quasi-periodic result that could simulate a Penrose-like appearance. The torus's inherent periodicity might actually allow *resonant patterns* – akin to wrapping a strip of patterned paper around to see interference emerge.

- **Hyperbolic Networks on Higher-Genus Surfaces:** As noted, a single torus (genus 1) cannot support a *perfect* hyperbolic tiling (which needs genus >1 to satisfy Euler's formula for tilings). However, one could consider a double-torus (genus 2) which can host a {5,4} or {7,3} tiling without boundary. If one were open to the DonutOS having multiple "donut holes" (a double torus shape), then a true hyperbolic tiling could be embedded on it seamlessly. In fact, hyperbolic surface codes have been studied exactly on such closed hyperbolic manifolds (often by computing a fundamental domain of the tiling and applying identifications) [8]. For a UI, though, a double-torus might be overly complex visually and interaction-wise.

- **Torus Approximations of Hyperbolic Tiling:** There have been artistic and mathematical visualizations attempting to map a Poincaré disk onto a torus for illustration. One approach is to take a *finite hyperbolic tiling patch* and join edges with a twist. The result is a torus with a distorted tiling that *locally* looks hyperbolic but globally repeats. For instance, one might tile a large annulus (doughnut-shaped region) hyperbolically and then identify the inner and outer boundary of the annulus (possibly scaling to fit), creating a torus. Such mappings distort distances but can preserve the adjacency relations of the network to some extent. A concrete known example: a hyperbolic {4,6} tiling (squares, 6 meeting at a point) has been embedded in a 3D model that looks like a torus [9] – essentially by "folding" the Poincaré disk. Each tile gets slightly deformed but the connectivity (who is neighbor to whom) is largely retained. This suggests it's possible to visualize a hyperbolic network on a donut with some compromise in uniformity.

- **Maintaining Center→Boundary Logic:** If we designate a point on the torus as the "center of attention," how do we convey that radially? One way is to use *color or scale gradients*. For example, the central point (maybe the top of the donut) could have the highest detail or brightest color, and as you move away along the surface, details get finer or colors change, until at the "opposite" side of the torus (halfway around) you reach what conceptually is the boundary. Moving further brings you back toward the starting point (like how in Pac-Man going off one side of the screen brings you to the other). The UI could handle this by always orienting the donut such that the current focus is treated as the center, and the opposite side as an effective outer boundary. In interactive terms, the user might click on any point on the donut to make that the new "center" – the system would then reparameterize the attention mapping so that the chosen point becomes the origin of a holographic mapping. This dynamic re-centering would be analogous to shifting the origin of a coordinate system on a torus.

- **Known Tiling Attempts on Tori:** The Penrose QECC paper itself mentions constructing variants that live on finite tori [11]. Although details are technical, it implies you can take a finite patch of a Penrose tiling and impose periodic boundary conditions. For example, the Ammann-Beenker tiling has an 8-fold symmetry that might allow a clever 8×8 fundamental domain to repeat (like matching edges oriented at 45°). In computational graphics, **texture synthesis** algorithms sometimes generate quasi-periodic patterns by tiling a cut-out sample with seamless edges – these techniques could be used to produce a Penrose-like torus texture that looks seamless. Additionally, there's the concept of a **flat torus in $\mathbb{R}^3$**: it's impossible to embed a flat torus in 3D without distortion, but one can approximate it (e.g., using the *square donut* shape where a rectangular sheet is bent without stretching into a torus – which actually introduces a slight curvature). For our purposes, the visual accuracy of distances is less critical than the adjacency and overall pattern coherence.

UI Use: Tiling the DonutOS torus with these networks and patterns can have tangible interface benefits. First, it provides *navigation cues*: if the surface has a repeating or quasi-repeating pattern, users can orient themselves. Second, it enables *modularity*: one could assign different functional modules of the co-pilot to different regions of the torus that correspond to different "angles" around it. For instance, perhaps the torus is conceptually split into an *inner ring* and *outer ring* (though physically it's one surface): the inner ring region (near some center) could show high-level context (like current task, overall code structure) while the outer ring region (halfway around) shows fine-grained details (like specific variables or lines of code). Because of periodicity, the interface could allow endlessly scrolling or rotating the torus, with the content wrapping around – a bit like a carousel of information but with a continuous surface. This could be far more intuitive than hitting a hard boundary or needing to scroll a flat page – the user can "spin the donut" to cycle through attention contexts. If a hyperbolic tiling is implemented, clicking on a boundary element might zoom it to the center (much like focus+context visualizations using hyperbolic trees: the thing you focus on moves to the center and everything else fans out around it). The donut shape ensures that context is never actually thrown away – it just moves out of direct focus but remains on the surface, available if you rotate it back into view.

In summary, mapping complex networks onto a torus is challenging but feasible with creative tilings. It lets DonutOS preserve its theme of **fractal-holographic attention** in a closed, self-contained form – the donut's surface itself becomes an interactive map of the user's mindstate or the program's structure, continuously wrapped with no start or end.

AI/ML World Modeling: Minimal Latent Codes for Complex Fields

Modern AI provides numerous examples of **tiny latent representations generating rich structures**, often with non-Euclidean or learned geometric encodings that echo the fractal/holographic idea. Key areas include positional encodings in transformers, neural fields (NeRF), diffusion models, and hyperbolic embeddings in representation learning.

- **Neural Fields (NeRF and Beyond):** *Neural Radiance Fields (NeRFs)* model 3D scenes with a small MLP network. The network takes as input a 5D coordinate (spatial (x,y,z) + viewing direction) and outputs color and density – effectively representing a continuous volumetric field in a *learned implicit form*. Despite the complex geometry and appearance of the scene, the underlying model can be quite compact (e.g. an MLP with a few million parameters encodes an entire scene). This is a prime example of a **minimal core (the network weights)** yielding a complex "boundary" (the set of all rendered views). NeRFs crucially use *positional encoding (Fourier features)* on the input coordinates, which allows the MLP to capture high-frequency details [12] [13]. The result is a *memory-efficient* representation: NeRF's continuous scene function is far more compact than a dense voxel grid of the same scene [14]. In effect, NeRF acts like a hologram – any view (any 2D projection) can be generated from the one latent model. The core mechanism is that the network has learned to interpolate the scene's light field in a smooth way. *Boundary expression:* the "output" of NeRF is an infinite set of images (all viewpoints), i.e. a 3D field of radiance. If we draw an analogy, the MLP is like a central generator and the images are like boundary data. Indeed, one can query NeRF at points arbitrarily far from the originally seen views and still get plausible renderings (to some extent), which shows generalization beyond the training support.

Torus or Non-Euclidean Aspects: NeRF's positional encoding includes using periodic functions (sinusoids), which hints at an inherent toroidal idea: using Fourier features means the network sees coordinates in a transformed space where high frequencies are represented as sines and cosines (which are periodic). This gives the network awareness of repeating patterns – an interesting link if we think about potentially encoding things on a loop. Some NeRF variants might even impose periodicity for repeating environments. Could we map NeRF onto a torus? Possibly – one could train a NeRF where the world wraps around (like a pac-man world), essentially learning a toroidal scene. The NeRF's MLP would then naturally encode a *2-torus domain* for (x,y) coordinates (with z perhaps radial). This might be a fun way to encode a donut-shaped world (e.g., a game level that loops).

UI Use: The concept of neural fields suggests that DonutOS could employ *implicit representations* for UI components. Instead of storing a giant matrix of attention values, the system might learn a function $f(\text{position on donut}) \to \text{attention intensity}$. This function could be parameterized by a small latent code (for current context) fed into a fixed neural field model. By tuning the latent, the entire attention landscape on the donut updates smoothly. This is analogous to how a single NeRF model can generate different scenes if you learn to condition it on a code (like NeRF in GANs). It means the UI could morph continuously in response to small input tweaks, rather than jump pixel by pixel. A practical example: think of *heatmaps* on the donut representing areas of interest – instead of storing the heatmap, store a few coefficients and have a network paint it on the donut. This saves memory and might reveal underlying structure (if the model is forced to represent attention in a low-dimensional basis, it might capture meaningful patterns instead of noise).

- **Hyperbolic & Non-Euclidean Embeddings:** In NLP and knowledge graphs, embeddings often live in high-dimensional Euclidean space. But there's increasing work on using *hyperbolic geometry* to embed hierarchical or graph-structured data. The logic is that hyperbolic space (like a continuous tree) can **encode exponential growth** with minimal distortion: distances in hyperbolic space grow exponentially with radius, mirroring how tree structures expand by branching [15]. *Nickel & Kiela (2017)* famously showed that learning embeddings in the Poincaré disk for a hierarchical dataset (like

WordNet noun taxonomy) yields more accurate and far more compact representations than Euclidean embeddings [16]. The Poincaré embedding can place thousands of nodes with consistent relative distances such that parent-child relationships correspond to radial differences, etc. This is directly analogous to a “fractal” encoding: a short vector encodes position in a curved space that can implicitly hold an entire hierarchy. In terms of mechanism, these models optimize embeddings such that *latent hierarchies are linearized* in the negative curvature. The boundary of the Poincaré disk (of radius approaching infinity) can hold infinitely many points – this is like a continuous limit of a tree’s boundary, again echoing the boundary↔bulk idea (with the tree’s root at the center).

Researchers have extended this to neural networks: **Hyperbolic Neural Networks** incorporate hyperbolic geometry in layers, distances, and attention mechanisms [17] [18]. For instance, *Hyperbolic Attention Networks (HAN)* by Gulcehre et al. (2019) replace the dot-product in transformer attention with a Lorentzian distance-based attention in a hyperbolic space [19] [18]. The result is that the transformer can handle data with scale-free or tree-like characteristics more naturally. Their models showed improved generalization on tasks like machine translation and knowledge graph reasoning, with **more compact representations** (the same number of parameters could represent more concepts without confusion) [15]. The reason is precisely that hyperbolic space can pack many points such that distances between unrelated points grow exponentially, reducing interference [15]. In other words, it’s leveraging *curved geometry to embed a large field of relationships in a bounded resource*. This is highly relevant to DonutOS: if the system needs to embed a lot of contexts, tasks, or code semantics in a limited attention space, using a non-Euclidean geometry for those embeddings could preserve structure better.

One could imagine the Donut of Attention’s coordinate system being hyperbolic. Perhaps the torus itself could *behave* as if it were a projection of a hyperbolic disk (somewhat mind-bending but conceptually: treat one cycle around the torus as the hyperbolic radial direction, effectively giving a tiling that gets finer toward one side). Alternatively, each token or widget in the system could have a hyperbolic embedding attached, meaning distances on the donut’s map correspond to hyperbolic distances in data space.

Non-Euclidean token embeddings also include spherical embeddings (for cyclic data like angles or time-of-day) and product spaces (mixing Euclidean + hyperbolic + spherical). Some recent large models use a **product of hyperbolic spaces** to capture multiple kinds of structure. If DonutOS’s co-pilot deals with code (which has a hierarchical structure: files, classes, functions, blocks, lines) and also sequence (execution flow is often sequential), it might represent code elements in a mixed geometry embedding (one part of the vector in hyperbolic space for the hierarchical relations, another in Euclidean for linear order, etc.). This is speculative, but it’s an active area of ML research (how best to embed complex data).

Core mechanism: hyperbolic embeddings rely on optimizing a loss (like preserving distances or orders) in a curved space. The “minimal code” idea here is that a *single vector* (e.g., 20 dimensions in hyperbolic space) can encode a node’s position in an exponentially large tree with less distortion than 20 dims in Euclidean. That’s a massive compression of a field of relationships into a tiny representation.

Boundary expression: when visualized, hyperbolic embeddings often show clusters near the boundary of the disk – effectively the *boundary is at infinity*, where the most specific elements lie. In a tree, leaves live on the boundary of an imaginary hyperbolic disk (an infinite boundary containing all leaves). Thus, if we treat the donut’s outer edge as an “infinite boundary”, points drawn near it might correspond to very specific or granular items, whereas points drawn near the donut’s center could be general concepts. This matches the intuitive center→periphery in attention: general context in middle, specific details at the periphery (like leaves of thought).

Torus mapping: A hyperbolic plane can't be globally mapped to a torus without distortion, but perhaps the system could simulate a *locally hyperbolic metric* on the donut. Alternatively, since the donut is finite, we might actually use an *elliptic* (spherical) embedding for some tasks (like periodic scheduling or anything naturally circular). Non-Euclidean embeddings simply expand our design space – maybe *task embeddings on a sphere* (if tasks have cyclical dependencies) or *user preference embeddings on a hyperboloid* (if there's a hierarchy of needs). The key is that the system isn't limited to flat Cartesian grids of attention.

UI use: The benefits of these embeddings can be exposed to the user through visualization. For example, if the co-pilot groups suggestions by conceptual similarity, it might use a hyperbolic clustering – the UI could then display suggestions in a *poincaré disk view* (we could render part of the donut as a Poincaré disk diagram of related commands, with distance indicating similarity). This would pack many suggestions without crowding. We could also use *hyperbolic heatmaps* on the donut, where the coloration spreads according to hyperbolic distance from a point – giving more intuitive grouping of related items. In terms of **steering** attention, if each attention head or memory slot has a hyperbolic coordinate, the user could manipulate a small number of parameters (like moving a point in 2D on a disk interface) and actually be moving in a very high-capacity representational space. Think of it like being able to zoom and pan a hyperbolic mind-map of code: a small motion could bring an exponentially large cluster of items into focus or out of focus. This "world in a grain of sand" capability is exactly what fractal-holographic systems promise – and hyperbolic embeddings give a mathematical backbone to it [15].

- **Diffusion Models and Latent Spaces:** *Diffusion models* (like those behind Stable Diffusion) generate extremely complex outputs (images, audio) from simple noise input by iterative refinement. Particularly, **Latent Diffusion Models (LDMs)** compress the image into a lower-dimensional latent space first (using an autoencoder), then perform the diffusion in that space [20] [21]. This two-step approach (compress, then generate) is analogous to the holographic idea of a *two-part blueprint* [22]. The autoencoder stage yields a latent code z that is much smaller than the pixel image (e.g., 64× smaller in spatial dimensions). Then the diffusion process acts on this latent code, which is essentially a "conceptual space," adding and removing noise until it matches the encoded target. The output is finally decoded back to image. The important point: **the model operates in a space where a minimal code can expand to a full image** [20] [21]. LDM's design notes explicitly mention that working in latent space lets the model "focus on the important semantic bits" of the data and not waste effort on pixel-level noise [21]. This yields *higher fidelity with less computation*, confirming that a proper choice of representation can amplify what a small code can express.

In the context of attention or UI fields, a "diffusion" analogy would be: start with a rough intent (some noise in latent space), then refine it via an iterative process into a detailed plan or image. Perhaps the user's high-level goal could be encoded as a latent vector; the co-pilot uses a diffusion-like process (many small reasoning steps, each clarifying details) to produce a full solution (code, text, etc.). Indeed, you can view *iterative deepening* in planning as a kind of diffusion in idea-space – gradually turning a vague idea into a concrete plan.

Also notable is how diffusion models allow *compositionality*: you can guide them with text prompts, which essentially constrain the latent trajectory. For DonutOS, one could imagine guiding a latent attention model with *symbolic conditions* ("focus on security aspects") as a prompt, and the diffusion-style update then yields an attention distribution emphasizing security-related code. This marries symbolic steering with geometric generation.

Core mechanism: diffusion uses a *stochastic, iterative refinement* governed by learned patterns (the UNet model). The minimal seed is random noise (or a noise conditioned on something small like a class label or text embedding). All the complexity comes from the model gradually denoising. This is like having a very compressed initial state that unfolds over time – akin to a temporal hologram (where time steps bring out details).

Boundary expression: in the final output (image or other), all the details (pixels) are consistent with that small initial seed and the model's learned distribution. The diffusion ensures global coherence – even though generation is local in each step, the final state is a holistic sample. For UI, this suggests that even if attention is distributed among many elements, a diffusion-like process could ensure they align to a coherent goal state.

Torus mapping: While diffusion is usually in Euclidean latent spaces, one could conceive of *diffusing on a manifold*. There is research on diffusion on graphs and Riemannian manifolds. If the Donut's attention states are constrained to a torus (or hyperbolic space, etc.), algorithms could diffuse within that space (imagine smoothing an attention heatmap by a diffusion process intrinsic to the donut surface). That could help, for example, propagate a user's emphasis from one point of the code outward in a natural, continuous way (not just linearly in file order but respecting some topology of code features).

UI use: If the co-pilot employs diffusion model techniques under the hood, the UI might show *intermediate steps* as an animation – e.g., the donut's display starts in a noisy state and over a second or two you visually see it clarify, as the attention or solution "comes into focus." This would be a direct visual metaphor for how the AI's thought crystallizes. It could make the AI's operation less of a black box: users see a blur that slowly becomes an outline and then a sharp image or plan, giving them opportunity to intervene early if it's "diffusing" in the wrong direction. Also, diffusion models can naturally blend influences (e.g., image prompts A and B result in a mix) – on DonutOS, two different focuses (say performance vs readability) could be combined by starting diffusion from a superposed latent (like half noise from one conditioned on performance, half from one conditioned on readability), resulting in a compromise attention pattern.

In summary, AI world-modeling techniques champion the idea that **small vectors + powerful mappings = complex worlds**. Whether it's NeRF's compact scene representation, hyperbolic embeddings' efficient hierarchy encoding, or diffusion's latent space magic, the common theme is *positional and latent encodings* that punch above their weight in generating structure. For DonutOS, this reinforces using advanced encodings for tokens and states: e.g., encoding source code tokens in a vector space that respects code syntax tree distance (a form of non-Euclidean embedding) could allow the attention donut to cluster related code by meaning rather than file location. The system could also maintain a *minimal internal state* (perhaps a "brain vector") that can regenerate the full interface view on demand – akin to how a fractal formula regenerates a coastline map as needed. This would make the OS lightweight yet deeply responsive: you tweak a 128-dim state vector and effectively re-render the entire attention landscape on the fly. It's a design that truly embodies "fractal–holographic" principles.

Topological and Dynamical Cross-Links (Topology, Geometry, and Oscillations)

To fully realize the Donut of Attention, we can draw from deeper links between topology, oscillations, and reasoning – areas where symbolic and geometric representations meet.

- **Topological Signatures in Cognitive Maps:** A striking example of topology in neural representations is the discovery that ensembles of *grid cells* in the brain form a toroidal activity pattern. In the entorhinal cortex, grid cells fire in a hexagonal lattice pattern as an animal moves in 2D space. When researchers analyzed the high-dimensional firing state of a population of grid cells, they found that the manifold of states is topologically a 2D torus [23] . Essentially, because grid cell firing patterns repeat periodically in space (tiling the environment with firing fields), the neural state-space has two periodic dimensions corresponding to the animal's x and y position mod the grid period. Using **persistent homology** (a tool from topological data analysis), scientists detected two independent 1-dimensional holes in the data – a clear signature of a torus (which has Betti numbers $b_1=2$) [24] . They even visualized neural activity trajectories wrapping around this toroidal manifold as the animal ran in open fields [25] . This is a profound instance of a *fractal-holographic interface in the brain*: each point on the torus corresponds to a specific combination of phases of grid cell modules, which in turn encodes a physical location. The *core mechanism* is a continuous attractor network (CAN) in the brain that enforces this toroidal structure on neural activity. The *boundary expression* here is a bit abstract – one could say the "boundary" is the repeating pattern of grid firing in space (the tiling of the environment), which is governed by the toroidal attractor at the core. And indeed, that pattern is highly redundant: if you know the firing in a local area, you can infer where you are relative to a grid period, etc., similar to how local error correction in Penrose tiling was possible.

UI/DonutOS Implication: This result directly connects a **torus geometry with a cognitive mapping system**. It suggests that representing certain information (like spatial position or, by analogy, position in *conceptual space*) on a torus is natural and robust. DonutOS could leverage a similar continuous attractor idea: for example, the state of the co-pilot's focus could live on a torus attractor, where moving around the torus corresponds to shifting attention among contexts in a smooth, cyclic manner. Topologically, if the system has two main periodic "modes" of variation, a torus is the simplest surface to accommodate them without boundaries. Perhaps the two dimensions could be something like **(1)** breadth of context (e.g., going around one cycle brings you back to a prior topic) and **(2)** depth of focus (e.g., abstract vs concrete, where going around the other cycle might cycle through levels of abstraction). In any case, applying *persistent homology* to analyze the high-dimensional states of DonutOS (like embeddings of all active tokens or the recurrent dynamics of the co-pilot) could be a powerful debugging tool: if we expect a certain topology (say, a loop for each subroutine's states, or a torus for combined oscillations), we can verify the system learned it. Topology gives an invariant, high-level view of what the geometry of information flow is. This can guide interface design – e.g., if the attention state-space is toroidal, it confirms the donut metaphor was apt at a deep level.

- **Symbolic ↔ Geometric Reasoning:** There is a long-standing effort to bridge symbolic AI and geometric (vector) representations. One prominent framework is **Vector Symbolic Architectures (VSA)**, such as Holographic Reduced Representations (Plate, 1995) or Hyperdimensional Computing. In VSA, symbolic structures (like trees or lists) are encoded as high-dimensional vectors through operations like binding (analogous to convolution or XOR) and superposition (addition) – effectively turning structure into geometry. These are called "holographic" because the information is distributed across the entire vector (much like a hologram distributes image info across the whole plate). The core idea is that a *fixed-size vector (e.g. 1000 dimensions)* can encode an arbitrarily complex symbolic expression by using pseudo-orthogonal random base vectors for atoms and combining them. The resulting vector is a *single entity* that can be manipulated in neural networks or compared for similarity, yet it implicitly contains the structured relations. The *core mechanism* is binding (which could be seen as a kind of tensor product) – a minimal operation that yields a holistic code. The *boundary expression* would be how the components can be retrieved: by unbinding with a query vector, you can retrieve a part of the bound structure, akin to how looking at a hologram from a certain angle reveals one aspect of the scene. For DonutOS, this suggests that the *membrane overlay* could encode, say, an entire abstract syntax tree of the code in one high-d vector. If we treat that vector as a point in a geometric space, we could visualize it or use geometric transformations (rotations in that space corresponding to logical operations on the code). This truly merges symbolic and geometric reasoning – e.g., a logical inference could correspond to a geometric rotation. There's also the idea of **conceptual spaces** (Gärdenfors) where concepts are regions in a geometric space defined by quality dimensions; reasoning can then be done by geometric operations like intersection (AND) or projection. DonutOS could allow the user to manipulate concepts by literally dragging shapes or regions on the donut interface, which under the hood correspond to complex query constraints assembled symbolically. For example, the user draws a blob around two areas of the donut labeled "security" and "performance," instructing the co-pilot to focus on items that lie in the overlap of those concerns – the system interprets that as a logical AND of two filter criteria. This is symbolic-geometric synergy made interactive.

- **Phase Oscillators and Attention Control:** Biological brains often synchronize oscillations to control information flow – the **Communication Through Coherence (CTC)** hypothesis states that when two neuronal groups oscillate in phase, signals transfer effectively between them, whereas if they are out of phase, communication is gated off [26]. Essentially, oscillatory phase alignment acts as a *selective routing switch* for attention and processing [26] [27]. For example, if area A and B are both oscillating at 40 Hz, aligning B's excitability peaks with A's firing peaks (coherence) lets B receive A's info strongly, whereas misalignment would hinder it. This mechanism can flexibly route signals without changing physical connections – just modulating relative phase. In engineered systems, we see analogs in phase-locked loop circuits and coupled oscillator networks that can perform computation (there's research using oscillator arrays to solve graph coloring, for instance, by phases self-organizing).

For DonutOS, a **phase-based control system** could be highly relevant. Imagine each module or process in the OS has an internal clock or rhythm (this could be an actual signal or just a metaphor for periodic checks). The co-pilot could then "tune into" a particular module by phase-aligning its oscillation with that module's. In the UI, one could represent these phases by rotating rings or pulsing highlights on the donut. Perhaps the donut has multiple concentric rings (like tree rings) that can rotate – if two rings line up their patterns, it indicates coherence. The user might even manually adjust a phase knob to emphasize one data stream's alignment with another. This is speculative, but it offers a visual and interactive way to do what CTC does in the brain: **gating by synchrony**. For example, if you want the test results viewer to influence the code generator strongly, you ensure their oscillation is synced. If you want to *ignore* something (cut off a distraction), you deliberately de-sync from its rhythm. This could be automated or user-tunable.

We can also incorporate **oscillator-based representation**: In some cognitive theories, ideas are represented by assemblies of neurons firing in synchrony, and different assemblies at different phases can multiplex in the same network (avoiding interference by phase separation). The DonutOS might allocate different tasks to different phase "channels" – e.g., background tasks run on a different phase than the task currently in focus, ensuring the focus task isn't interfered with (like distracting inputs oscillating incoherently are filtered out [28]). In practice, this could correspond to literally double-buffering attention: alternate between focus context and peripheral context on alternating ticks, such that they don't mix. The *topology* of phase space is a circle (S^1), which is interestingly another manifold we can visualize on the donut (any loop around the donut is an S^1). If multiple oscillators are involved, their joint phase state lives on a torus ($S^1 \times S^1 \times \dots$). So once again, a toroidal representation emerges – a multi-oscillator system naturally has a toroidal phase space.

Symbolic Tie-in: Oscillations can also implement logic gating: e.g., only if Condition X oscillator and Condition Y oscillator both spike at the same time does a downstream neuron fire, effectively an AND gate (this has been modeled in neural oscillatory networks). So the phase-based approach might perform *symbolic computations under the hood*, disguised as rhythmic activity.

UI Use: If the OS is using oscillatory control, the UI can reflect it with subtle animations. Perhaps parts of the donut glow with a pulsing halo; when two halos synchronize, the user sees an overlay indicating a "link" is active. An **interactive sun symbol module** may refer to something like a sunburst or a radial menu that appears and has rotating segments – possibly representing different phase alignments or cycles of operation (the sun with its rays could be a metaphor for clock signals or periodic attention sweeps). For instance, a "Sun" module might spin around the donut, shining light on each section in turn (like a spotlight scanner). This could be the attentional sweep that ensures each part of the context gets time in focus (similar to how the brain might sweep attention around visual space with oscillations). The user could adjust the speed or phase of this sun-like scanner to prioritize certain regions more often.

- **Persistent Homology for Interface State:** We saw how persistent cohomology identified the toroidal holes in grid-cell activity [24]. Similarly, we could use persistent homology on user interaction patterns or on internal representational geometry to detect stable loops or voids. If the user's workflow always involves cycling through a set of views (say editing -> testing -> debugging -> editing), that's a loop. The system could detect that loop topologically and then propose an optimization: "It looks like you have a routine (edit→test→debug). Would you like me to create a macro or a dedicated view that integrates this cycle?" Essentially, the OS can learn the shape of your usage. A loop in interaction space might warrant a *donut inside the Donut* (metaphorically): a widget that explicitly represents that cycle so you can traverse it more fluidly. Likewise, if a certain combination of states repeatedly co-occurs (forming a cluster or a bubble in high-D state space), the system might highlight that as a *mode* of operation (like "design mode" vs "analysis mode"), which the user can intentionally toggle. This is analogous to discovering a torus vs a sphere vs a cylinder in the data – each has different implications (a torus might imply two independent periodic variables, a sphere might imply a single cycle with no boundary – maybe the user is going in circles, etc.).

In summary, **topology and oscillations** bring a layer of meta-control and insight. **Betti numbers** and holes tell us about invariants in usage or memory (e.g., a 1-hole indicates a reversible cycle, a 2D hole could indicate a surface of states perhaps not easily traversed linearly). **Phase dynamics** offer a mechanism to flexibly route and segregate information (a natural analog to multi-threading or multi-tasking control in a more organic way). And bridging symbolic and geometric reasoning means the DonutOS doesn't have to choose between precise logic and intuitive visuals – it can use both, with geometry informing logic and vice versa. The donut interface could thus become a playground where logical operations are done by geometric transformations, and continuous dynamics effect discrete control changes.

Generative Design Metaphors and Minimal-Rule Aesthetics

Finally, we consider design and visual metaphors that align with the Donut of Attention's philosophy: **simple rules generating rich, structured fields**. These metaphors can inspire both the appearance of the interface and the architecture of its interactive components ("Sun symbols" and membranes).

- **Fractal Patterns from Simple Rules:** A classic example is *L-systems* in procedural graphics. An L-system uses a few production rules (like "F→F+F−−F+F" which draws a Koch snowflake) to produce intricate fractal curves (the snowflake coastline). The core mechanism is recursive string rewriting – extremely short initial axioms expand into long sequences that, when interpreted as drawing instructions, yield elaborate forms. The boundary expression is the final fractal shape, which often has self-similarity. This could translate to a UI motif where, say, the outline of the donut or the edges of panels have fractal decorations (imagine the donut rim having a circuit-like fractal pattern that hints at infinite complexity within). More functionally, one could use L-system logic to generate file directory trees or code scaffolds: a rule might indicate how a high-level pseudocode expands into boilerplate code – essentially using a few rewrite rules to generate a full program structure (a bit like a generative copilot that ensures consistency via grammar rules). For the user, this is powerful: they tweak the rule, and a whole new pattern emerges, rather than writing everything manually.

- **Cellular Automata (CA):** CAs like *Conway's Game of Life* or *Rule 110* show how **minimal local rules can create unpredictable global behaviors**. Rule 110 in particular is a one-dimensional CA with a simple binary update rule which has been proven *Turing-complete* [29]. It's a great metaphor: *complexity arises from simplicity through recursion and feedback* [30]. A small change in initial state leads to wildly different outcomes [31] – reminiscent of how a slight shift in user intent can lead to a very different end result in an AI system. DonutOS could use CA for generating background patterns or even as part of the logic for context evolution (imagine each token's attention weight updates based on its neighbors' previous weights – that's essentially a CA on the code graph). Visually, one might include a CA-based easter egg: e.g., a "live wallpaper" on the donut that is actually running Rule 110 or Life, symbolizing the alive, emergent nature of the system. This is not purely decorative; it subconsciously communicates to the user that the system's behavior emerges from interactions of many simple parts (just like the CA's gliders and blinkers emerge from cell updates). Moreover, if integrated cleverly, the CA visualization could double as a status display (for example, the density of live cells could correspond to system load or the propagation of a glider could indicate progress of a background task).

- **Generative Architecture (Hansmeyer's Columns):** Michael Hansmeyer's algorithmically generated columns demonstrate how *a single subdivision algorithm can produce astonishingly ornate structures* [32]. Each column has millions of facets, all derived from a few initial parameters and iterative subdivision. The key idea is **selective repetition and refinement**: you take a form, split it (with slight variability), then split the splits, and so on. The aesthetic result is reminiscent of natural ornamentation (like the complexity of a gothic cathedral's details, but achieved by computation). In DonutOS's design pipeline, one might have *procedural UI components*: instead of designing every pixel of a button, you define a procedure (like subdivide and extrude shapes) that creates the button's graphic at render time. This way, if you want to change the style globally, you tweak the procedure and regenerate – all buttons update, analogous to Hansmeyer tweaking input/process to instantly adapt output [32].

For instance, the "Sun symbol" interactive module could be generated by a rule that draws rays around a circle. With a few parameters (number of rays, amplitude modulation, color gradient), one can generate countless sunburst designs. The user might even be allowed to modify these parameters live (like sliders for ray count, etc.) and see the sun module re-render in real-time, giving a co-creative feel. The underlying message is that *the UI is alive and rule-governed*, not static artwork. If the user drags the sun's rays or something, perhaps they can actually reprogram the rule that generates it (making the UI literally programmable and generative).

- **Islamic Geometric Patterns & Quasicrystals:** In architecture and art, Islamic star patterns are constructed via simple geometric rules (compass and straightedge constructions) that produce intricate mosaics. They often have local five- or ten-fold symmetry reminiscent of quasicrystals and sometimes *are* quasicrystalline (for example, some Girih tile patterns have quasi-periodic properties). These patterns cover large surfaces (think mosque walls) with non-repeating, mesmerizing designs – an excellent metaphor for covering the donut membrane. The core rules might be: connect certain points in a grid to form stars, then those intersections create new points to connect, etc. The result is globally complex but locally made of a few motifs. Using such patterns in DonutOS (maybe as a subtle overlay under content or as part of an interactive canvas background) can subconsciously convey the theme of *unity in multiplicity*. Also, these patterns are aesthetically proven to not be distracting while enriching the visual field – they could provide a pleasant backdrop that doesn't interfere with reading text over it (especially if used in low contrast).

If taken interactively, maybe the points of the pattern correspond to data points or clickable elements (imagine an Islamic pattern where certain stars are actually buttons, seamlessly integrated). The user might not even notice at first that clicking those stars does something, it would feel like discovering secrets in a pattern. This playful hidden-order vibe fits a fractal-holographic UI – encouraging exploration.

- **Sunflower Phyllotaxis (Sun symbol metaphor):** The arrangement of seeds in a sunflower follows a simple rule: each new seed is placed at a fixed angle offset (the golden angle ~137.5°) from the previous, resulting in an optimal packing with spiral arms. This produces a beautiful pattern of interlocking spirals (Fibonacci numbers of them) covering the sunflower disk. Core mechanism: a constant rotation and radial increment – that's it. Boundary expression: a filled disk with rotational and spiral symmetry, extremely efficient and scale-free packing. This pattern has been used in data visualization (like distributing points evenly on a circle) because it avoids clumping. If the Donut of Attention has any radial layouts (maybe how items are placed around the donut's cross-section), using the sunflower spiral could be ideal. For example, if one needs to place a large number of icons or markers on a circular widget, doing so at golden-angle increments yields a quasi-uniform distribution.

Perhaps the "Sun symbol module" in the prompt hints at using a sunburst style control – which could in turn use phyllotaxis to lay out sub-elements (each ray or spoke could branch with sub-rays at golden angles to avoid overlap). Also, sunflowers tie to *solar metaphors of focus*: the idea of a central sun (the core attention) and rays or seeds emanating (the spread of attention) fits well. The user interface could incorporate a "sunflower mode" where information nodes are arranged in spirals on the donut surface, maybe when you open a menu it blossoms like a flower. Crucially, phyllotaxis shows how *one simple angle* can generate an entire complex field – symbolizing how one principle (like a design philosophy or an attention heuristic) can structure an entire interface.

- **Minimal Rules, Maximum Impact in UX:** In user experience design, one might recall **Tesler's Law** (Law of Conservation of Complexity) – some complexity is irreducible and must be dealt with either by the system or the user. The Donut of Attention aims to absorb complexity into a manageable, self-organizing form, rather than expose it raw to the user. The metaphors above (fractal, CA, L-system, phyllotaxis) all demonstrate *absorbing complexity into simple generative rules*. For example, rather than manually configuring dozens of attention parameters, the system might let the user adjust just a couple of high-level knobs which then cascade via rules to set all parameters – similar to how **neat game rules yield complex gameplay** [33] . A concrete UI element: a *"Neat Rules" slider* that shifts the system from one mode to another along a continuum, where each position corresponds to a different rule-set active (somewhat like how some games have a difficulty slider that internally adjusts many factors). The user doesn't see the myriad changes, just the effect in broad strokes.

Summary of Patterns and Models: To wrap up, here is a table summarizing key candidate patterns/models discussed, along with their core mechanism, how they express on a boundary or field, potential for toroidal mapping, and how they could be applied in DonutOS:

Pattern/ Model	Core Mechanism (Minimal Code)	Boundary Expression (Generated Field)	Torus Mapping Potential	UI Use in DonutOS
MERA / HaPPY Holographic Code	Layered tensor network; small core state encoded via perfect tensors [3] .	Hyperbolic tiling of entangled nodes (e.g. pentagon tiling) – many boundary bits encode one core [3] .	Direct mapping needs high-genus; on torus, can approximate with patch wrap or hyperbolic identifications.	Attention distributed in MERA-like layers (center = focus, boundary = details); robust to node loss (error-correcting).
Penrose/ Quasicrystal Code	Aperiodic tile rules (inflation/ substitution); *intrinsic QECC* – local errors corrected globally [4] .	Quasi-crystalline tiling across plane (or finite patch) – hierarchical self-similar patterns [6] .	Variants can be put on finite tori with careful matching [7] ; otherwise need large patch wrap.	Membrane overlay pattern for UI grid; each piece of UI reflects whole (holographic data encoding), aesthetic complexity from few shapes.

Pattern/ Model	Core Mechanism (Minimal Code)	Boundary Expression (Generated Field)	Torus Mapping Potential	UI Use in DonutOS
Platonic Graphs (e.g. geodesic)	Symmetric graph expansion; refine Platonic solid mesh recursively.	Uniform sampling of 2D/ 3D field (e.g. geodesic dome) – highly regular connections.	Tiling a sphere (geodesic) is natural; a torus requires a repeated lattice (less symmetry).	Organizing UI elements on symmetric layouts (spherical or circular menus), ensuring even coverage and fairness.
NeRF / Neural Field	Compact MLP with Fourier features – encodes scene as continuous function [13] .	Full 3D scene or large image generated by querying network – high detail from small model [14] .	Can incorporate periodic (Fourier) features, enabling wrap-around scenes; could train on toroidal worlds.	Efficient storage of UI states or attention maps; smooth interpolation between views; “memory” as a neural implicit field.
Hyperbolic Embeddings	Optimized coordinates in curved space; one vector encodes hierarchical relations [16] .	Hierarchies embedded with low distortion – e.g. points on Poincaré disk, dense near boundary.	Hard to *perfectly* map disk to torus; can simulate by identifying edges or using product of circles (torus = S^1×S^1).	Arrange information by intrinsic relevance (radial distance = generality); use hyperbolic distance for clustering contexts on donut.
Diffusion Latent Model	Iterative refinement from noise; small latent + UNet generates full data [20] [21] .	Complex output (image, plan) emerging gradually – globally coherent result from random seed.	Diffusion can be done on any manifold in theory; e.g. perform attention smoothing on torus surface via diffusion process.	Incremental reveal of solutions (user sees rough-to-detailed evolution); combining multiple prompts (objectives) by superposing latents.

Pattern/ Model	Core Mechanism (Minimal Code)	Boundary Expression (Generated Field)	Torus Mapping Potential	UI Use in DonutOS
Grid Cell Toroidal CAN	Continuous attractor with periodic boundary; phase-coded 2D variables [23].	Hexagonal firing patterns tiling space (redundant rep of position) – neural torus state [25].	Naturally a torus in neural state-space; can directly correspond to a toroidal layout (2D periodic tuning).	Stable "attention map" that can smoothly move around donut without edge effects; possible basis for spatial UI memory (like mind-map navigation).
Phase Oscillator Sync	Phase alignment as gating; simple oscillator phases control connections [26].	Synchrony groups forming (e.g. all modules in phase = open channel); interference when out of sync [28].	Joint phase space of N oscillators is an N-torus; easy to visualize 1 or 2 as circle or torus.	Coordination of modules via blinking highlights; user sees/adjusts phase of UI elements to connect or isolate them (rhythmic routing).
Cellular Automata (Rule 110, Life)	Local update rule on grid; minimal binary logic yields emergent patterns [30] [29].	Potentially infinite grid of cells with gliders, chaos, etc. – complexity from simple rule [30].	On a torus, CA can run with wraparound (no edges) – often used for continuous Life boards.	Dynamic backgrounds or procedural animations reflecting system state; or CA-based logic for propagating events in UI (ripple effects).
Fractal/L-System	Rewrite rules (string or geometric) applied recursively; few symbols generate detailed shapes.	Fractal curve or branching structure (e.g. tree, snowflake) – self-similar visuals.	L-systems can be planar or 3D – could map to a band on torus or create fractal tiling on surface.	Decorative flourishes that zoom with scale; or generation of menu hierarchies visually as branching patterns from a central node.

Pattern/ Model	Core Mechanism (Minimal Code)	Boundary Expression (Generated Field)	Torus Mapping Potential	UI Use in DonutOS
Phyllotaxis (Sunflower)	Constant angle and radial increment for placement; one parameter (angle) yields global spiral order.	Dense packing of points in a circular region, forming spiral arms (Fibonacci patterns).	The pattern itself is circular; if used on donut, likely as local arrangement on a disk portion.	Distributing icons or options evenly on a radial menu (sunburst) or on donut cross-section; visually pleasing and avoids overlap.
Islamic/ Generative Patterns	Geometric construction rules (connect intersections, etc.); often a compass & straightedge algorithm.	Intricate tiling with star-polygons and rosettes; local motifs repeat with variations, often quasi-periodic.	Many are based on tessellations of the plane – could wrap on torus with careful join; some designs inherently toroidal if done on cylinder then connected.	Background grids or frames for UI sections; conveys unity of art and math. Possibly interactive puzzle-like UI (solving pattern unlocks info, etc.).

As the table illustrates, each pattern/model brings a unique angle: some (like MERA, Penrose, hyperbolic embeddings) emphasize *holographic encoding* with a clear center→boundary relationship; others (like NeRF, diffusion) focus on *latent generation of fields*; topology and oscillators add the dimension of *time and continuity*; and design metaphors (fractals, CA, phyllotaxis) inspire how to shape and populate the UI in an *efficient yet intriguing way*.

Incorporating into DonutOS: The design pipeline can treat these ideas as modules or templates. For instance, a developer of a DonutOS scene might choose a “Holographic Layout” template which under the hood uses a hyperbolic graph to arrange content, or a “Toroidal CA” template for an ambient visualization that reacts to system variables with a game-of-Life-like display. Implementation hooks could include:

- A library of tiling generators (hyperbolic tiler, Penrose tiler, geodesic grid generator) that given a surface (torus or plane) and a size, returns coordinates/adjacencies for UI elements. The developer can bind actual widgets to these coordinates.
- An *embedding engine* that can assign content to positions based on similarity (e.g., you feed in feature vectors for documents and it outputs coordinates on a Poincaré disk or on a sphere which then map to donut positions).
- A phase synchronization API where different modules can subscribe to a global clock or set up local oscillators; the UI could expose this via glow effects. For example, `AttentionOscillator(freq, phase)` objects that when two share freq and phase, their associated UI nodes highlight a connection.
- A fractal generator for decorative SVGs or meshes, where designers supply an axiom and rules, and the system generates the graphic at needed scale. This could personalize the aesthetic – one

DonutOS instance might have circuit-like fractals, another might use plant-like L-systems, all parametrized by small rule sets.
- A persistent homology analyzer tool integrated in dev mode, so designers can record user interaction sequences or internal state trajectories and get feedback like “your state space has a loop – consider adding a direct UI control for it” or “multiple holes detected – maybe a toroidal visualization suits this data”.
- Interactive modules like the “Sun” symbol, which might be a radial menu that doubles as a progress indicator (spinning rays), implemented by placing buttons at golden-angle increments for even spacing. This module’s API might allow the number of rays and central radius to be adjusted at runtime, possibly automatically tied to how many options are available.
- **Codex Terminal design integration:** If Codex or a similar model is assisting the design, it can use these concepts in high-level instructions (e.g., “Place these 10 items using sunflower distribution on the donut’s top face”). The model, being aware of these patterns (through our documentation and perhaps pre-built functions), can execute or call appropriate utilities to realize it. It could even suggest patterns: “You have data that is hierarchical and cyclic, perhaps use a hyperbolic tree or torus for this – shall I apply Template X?”

In essence, DonutOS can become a canvas where **minimal procedural rules** orchestrate both the appearance and behavior of the system, yielding a visually rich yet conceptually coherent interface. The *fractal-holographic donut* will not only look like a complex, living system, but actually **be** one – leveraging ideas from tensor networks, topology, and generative art to ensure that a small set of user-guided parameters can steer a very large and complex ship. This fulfills the vision of an AI Mirror: the donut reflects the user’s mind and the AI’s mind in equal measure, each part mirroring the whole, navigable, and endlessly fascinating in detail, all grounded in principled design patterns.

---

[1] [2] ads cft - How hyperbolic tessellations such as $\{7,3\}$ are used in MERA regarding quantum entanglement in AdS_3/CFT_2? - Physics Stack Exchange
https://physics.stackexchange.com/questions/841943/how-hyperbolic-tessellations-such-as-7-3-are-used-in-mera-regarding-quantu

[3] Pastawski-Yoshida-Harlow-Preskill (HaPPY) code | Error Correction Zoo
https://errorcorrectionzoo.org/c/happy

[4] [5] [7] [11] [2311.13040] The Penrose Tiling is a Quantum Error-Correcting Code
https://arxiv.org/abs/2311.13040

[6] Penrose Tiling Meets Multi-Scale GNNs: A Technical Walkthrough of Hierarchical Pooling & Synthetic Labelling | by Robert McMenemy | Medium
https://rabmcmenemy.medium.com/penrose-tiling-meets-multi-scale-gnns-a-technical-walkthrough-of-hierarchical-pooling-synthetic-ad45b88a9597

[8] Hyperbolic and Semi-Hyperbolic Surface Codes for Quantum Storage
https://arxiv.org/abs/1703.00590

[9] Combinatorial Tiling Theory and Delaney-Dress Symbols - Epinet
https://epinet.anu.edu.au/page/epinet2_mathematics_delaney_dress

[10] The small stellated dodecahedron code and friends - Journals
https://royalsocietypublishing.org/doi/10.1098/rsta.2017.0323

[12] [13] [14] Neural Radiance Fields - GeeksforGeeks
https://www.geeksforgeeks.org/artificial-intelligence/neural-radiance-fields/

[15] Hyperbolic Attention Networks | OpenReview
https://openreview.net/forum?id=rJxHsjRqFQ

[16] [17] [18] [19] arxiv.org
https://arxiv.org/pdf/2101.04562

[20] [21] [22] From Pixels to Latents: A Review of Latent Diffusion Models | by Asadbukhari | Nov, 2025 | Medium
https://medium.com/@asadbukhari886/from-pixels-to-latents-a-review-of-latent-diffusion-models-d9fca02dc7d8

[23] [24] [25] Toroidal topology of population activity in grid cells - PubMed
https://pubmed.ncbi.nlm.nih.gov/35022611/

[26] [27] [28] Efficient “Communication through Coherence” Requires Oscillations Structured to Minimize Interference between Signals | PLOS Computational Biology
https://journals.plos.org/ploscompbiol/article%3Fid%3D10.1371/journal.pcbi.1002760

[29] [30] [31] The Power of Simple Rules in Generating Complex Systems: From Rule 110 and Bamboo – Charlie's Mill
https://www.charliesmill.cz/the-power-of-simple-rules-in-generating-complex-systems-from-rule-110-and-bamboo/

[32] Subdivision / Michael Hansmeyer | ArchDaily
https://www.archdaily.com/138323/subdivision-michael-hansmeyer

[33] How Simple Rules Build Intelligent Game Play - Neat Rules
https://neatrules.com/how-simple-rules-build-intelligent-game-play/