# A Geometric and Information-Theoretic Interpretation of LayerNorm, FFN, and MHA in Transformers

Submitted to 39th Conference on Neural Information Processing Systems (NeurIPS 2025). Do not distribute.

**Anonymous Author(s)**
Affiliation
Address
email

**Abstract**

The Transformer architecture has become the foundation of modern artificial intelligence, yet the first-principles reasoning for its design remains surprisingly shallow. Standard explanations—that Layer Normalization (LN) combats "internal covariate shift" or that Multi-Head Attention (MHA) enables parallel processing—are correct but incomplete, failing to explain the deep synergy between the architecture's components. This paper proposes a new, unified framework that reframes these components through a geometric and information-theoretic lens. We argue that the Transformer block operates on a `symmetrize-and-structure` principle. The process begins with LN acting as a **Geometric Stabilizer**, an isotropic operator that projects token representations onto a fixed, (`d_model-2`)-dimensional manifold, ensuring dynamic stability. This symmetric "canvas" is then processed by MHA, which we identify as an **Anisotropic Processor** that performs **Axis-Aligned Subspace Decomposition**. A profound implication of this design is that the network is strongly incentivized to **encode meaning in its vector axes**. This is followed by the Feed-Forward Network (FFN), which we frame as a **Manifold-based Information Filter** that performs a complexity-reducing deformation to select for relevant features. This unified geometric perspective provides a more powerful, first-principles understanding of the Transformer's effectiveness and stability. It culminates in a proposal for a fully trainable, anisotropic LayerNorm, which would make the `symmetrize-and-structure` cycle architecturally consistent and biologically plausible, opening a new frontier in model design.

## 1 Introduction

The Transformer architecture, introduced by Vaswani et al. (2017), has revolutionized natural language processing and beyond. Its success is largely attributed to the self-attention mechanism, which allows for the modeling of global dependencies in a highly parallelizable manner. However, the architecture's remarkable stability and performance also rely on two other ubiquitous components in every block: a Layer Normalization (LN) layer and a position-wise Feed-Forward Network (FFN).

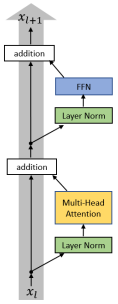


Figure 1: pre-LN Transformer Block

The conventional understanding of these components is largely functional and historical. Layer Normalization (Ba et al., 2016) is typically justified as a technique to stabilize training by normalizing the distribution of activations, thereby reducing "internal covariate shift." The FFN is explained as being necessary to provide non-linear expressive power, as a stack of linear attention layers would otherwise collapse into a single, less powerful linear transformation.

While these explanations are not incorrect, they are unsatisfying. They do not explain, for instance, the precise mathematical synergy between LayerNorm and the `sqrt(d_k)` scaling factor in attention. Nor do they provide a compelling rationale for why the specific form of the FFN—an expansion, a ReLU/GELU activation, and a contraction—is so effective. They describe *what* the components do at a surface level, but not *why* they are designed that way or what their deeper purpose is within the system's computational logic.

This paper introduces a new perspective that aims to fill these explanatory gaps. We propose a unified framework that interprets LN, MHA, and the FFN not as statistical or algebraic "tricks," but as fundamental operators performing geometric and information-theoretic transformations. Our core theses are:

1. **Layer Normalization is a Geometric Stabilizer:** Its primary role is to project token representations from the ambient $\mathbb{R}^{d_{\text{model}}}$ space onto a stable, lower-dimensional manifold. The fixed parameters of each LN layer define a unique, time-invariant geometric "canvas" for each processing step, ensuring dynamic stability.
2. **The Feed-Forward Network is an Information Filter:** Its role is to process the representations on this manifold. It does so by first orienting the manifold in a high-dimensional space and then applying a **complexity-reducing deformation** via the activation function, effectively filtering out irrelevant information before summarizing the result.
3. **Multi-Head Attention is an Anisotropic Processor:** Its role is to impose a learned, structured anisotropy on the symmetric space created by LayerNorm. By partitioning the feature space into specialized, parallel subspaces (the heads), it enables a "divide and conquer" strategy that forces the network to encode meaning in the very indices of its vector axes.

This framework provides a more rigorous, first-principles understanding of the Transformer. It explains the synergy between its components, provides a mechanism for its long-context stability, and offers a more powerful vocabulary for analyzing its internal workings.

## 2 Recap of pre-LN Transformer DecodeBlock

A Transformer `DecodeBlock` transforms an input sequence representation into an output sequence representation of the same size. While several variants exist, this paper exclusively analyzes the

**pre-LN** variant—where Layer Normalization is applied *before* the computational sub-layer—which is common in modern large language models.

- **Input to Block**: $X^{(L-1)} \in \mathbb{R}^{B \times T \times d_{\text{model}}}$. This is a batch of $B$ sequences, each with $T$ token vectors of dimension $d_{\text{model}}$.
- **Output of Block**: $X^{(L)} \in \mathbb{R}^{B \times T \times d_{\text{model}}}$.

The computation within the block proceeds as follows:

1. $X_{\text{norm1}} = \text{LayerNorm1}(X^{(L-1)})$
2. $Y = X^{(L-1)} + \text{MHA}(X_{\text{norm1}})$
3. $Y_{\text{norm2}} = \text{LayerNorm2}(Y)$
4. $X^{(L)} = Y + \text{FFN}(Y_{\text{norm2}})$

See *Addendum: DecodeBlock Pseudocode* for a step-by-step exposition.

## 3 The Geometric Stabilizer: A New Role for LayerNorm

We begin by re-examining Layer Normalization. In a standard pre-LN Transformer block, every computational sub-layer (both MHA and FFN) is preceded by an LN layer. We argue this is not merely for numerical stability, but to enforce a powerful geometric constraint on the state space.

### 3.1 The Geometry of Normalization: From $\mathbb{R}^{d_{\text{model}}}$ to a $(d_{\text{model}} - 2)$-Manifold

The LayerNorm operation on a single token vector $\mathbf{x} \in \mathbb{R}^{d_{\text{model}}}$ consists of two steps: a normalization and a learned affine transformation. Let us first analyze the normalization step:

$$\boldsymbol{z} = \frac{\boldsymbol{x} - \mu}{\sqrt{\sigma^2 + \epsilon}}$$

where $\mu$ and $\sigma^2$ are the mean and variance of the components of $\mathbf{x}$. This operation is not just a statistical rescaling; it is a geometric projection. The output vector $\mathbf{z}$ is not free to point anywhere in $\mathbb{R}^{d_{\text{model}}}$. It is constrained to lie on a specific, lower-dimensional surface defined by two properties:

1. **Zero Mean:** The sum of its components is zero: $\sum_{i=1}^{d_{\text{model}}} z_i = 0$. This equation defines a hyperplane in $\mathbb{R}^{d_{\text{model}}}$ that passes through the origin. This single linear constraint reduces the degrees of freedom by one.
2. **Unit Variance / Constant Norm:** The sum of the squares of its components is constant: $\sum_{i=1}^{d_{\text{model}}} z_i^2 = d_{\text{model}} \frac{\sigma^2}{\sigma^2+\epsilon} \approx d_{\text{model}}$. This equation confines the vector $\mathbf{z}$ to the surface of a hypersphere of radius $\sqrt{d_{\text{model}}}$.

The output of the normalization step, $\mathbf{z}$, must therefore lie on the intersection of a $(d_{\text{model}} - 1)$-dimensional hyperplane and a $(d_{\text{model}} - 1)$-dimensional hypersphere. This intersection is itself a sphere of one lower dimension (a $(d_{\text{model}} - 2)$-sphere), which is a manifold with $d_{\text{model}} - 2$ **intrinsic dimensions**.

The subsequent affine transformation, $\mathbf{y} = \boldsymbol{\gamma} \odot \mathbf{z} + \boldsymbol{\beta}$, where $\boldsymbol{\gamma}$ and $\boldsymbol{\beta}$ are learned parameter vectors, takes this perfect, standardized manifold and performs a layer-specific transformation. The scaling by $\boldsymbol{\gamma}$ warps the sphere into an ellipsoid, and the shift by $\boldsymbol{\beta}$ translates it. The final output $\mathbf{y}$ is thus constrained to a specific, learned, (`d_model - 2`)-dimensional ellipsoid-like manifold.

This projection onto a curved, lower-dimensional manifold has a profound computational consequence. While subsequent operations, such as the linear projections in MHA and FFN, are linear

with respect to the ambient $\mathbb{R}^{d_{\text{model}}}$ space, they act as powerful **non-linear transformations** on the intrinsic coordinates of the manifold itself. A simple linear shear in a high-dimensional space, for instance, can induce a complex, non-linear warping on the surface of a sphere embedded within it. This interplay between geometric projection and linear algebra is a pervasive and unappreciated source of the Transformer's non-linear expressive power, operating at every sub-layer of the network. For a concrete 3D illustration of this principle, see the *Addendum: A Geometric Analogy for Implicit Non-Linearity*.

### 3.2 Synergy with Attention: Aligning Representational Space with Computation

This geometric projection provides a profound explanation for the synergy between LayerNorm and the attention mechanism.

1. **Factoring out Magnitude:** By forcing all token vectors onto a bounded manifold, LN effectively factors out the vector's magnitude as a primary information carrier. A vector's "meaning" is no longer encoded in its length but in its **direction**—its specific location on the manifold.
2. **Congruence with Attention:** The core operation of attention is the dot product, `score = q · k`. We know that `q · k = ||q|| ||k|| cos(theta)`. Since the vectors used to generate `q` and `k` originate from the LN manifold, their norms are constrained. Consequently, the attention score becomes primarily a function of `cos(theta)`, the cosine similarity.

This reveals a deep design principle: **LayerNorm shapes the latent space to be hyperspherical, and Attention is an operator that naturally measures similarity on a sphere.** The architecture imposes a geometric structure on its data that is perfectly congruent with its primary computational operator. This also provides a first-principles explanation for the `sqrt(d_k)` scaling factor. The unit-variance property enforced by LN on the vector components is precisely what causes the dot product's variance to grow to $d_k$. The scaling factor is the exact mathematical antidote required to re-normalize the variance of the scores to 1, ensuring stable inputs to the softmax function. This rigorous causal link moves beyond the original explanation in Vaswani et al. (2017), which noted the effect without providing a mechanistic origin in the preceding normalization step.

**A Note on Metrics: Geodesic vs. Euclidean:** This alignment between geometry and computation becomes even more profound when we consider the concept of distance. In the high-dimensional Euclidean space, distance is a noisy and often uninformative metric. However, by projecting vectors onto a unit hypersphere, the model creates a space where the most meaningful measure of separation is the **geodesic distance**—the shortest path between two points along the curved surface of the sphere.

For two unit vectors on a hypersphere, this geodesic distance is simply the angle, $\theta$, between them. This angle is directly and linearly recoverable from the dot product that attention computes: $\theta = \arccos(\mathbf{q} \cdot \mathbf{k})$. This is a crucial insight. It means the model has discovered a geometry where its fundamental algebraic operation (the dot product) is a direct, uncorrupted measure of true geometric distance. The "curse of dimensionality" is sidestepped by adopting a geometry where angular separation *is* distance, perfectly aligning the representational space with the computational mechanism.

### 3.3 The Fixed Manifold: A Mechanism for Dynamic Stability

The geometric view of Layer Normalization provides its most significant explanatory power when we consider the dynamics of autoregressive generation. A key challenge for any generative model

is maintaining coherence and stability over long sequences. The "Geometric Stabilizer" framework reveals a powerful mechanism by which the Transformer achieves this, though the precise nature of the mechanism depends on the model's operational context.

The core principle is that the `gamma` ($\gamma$) and `beta` ($\beta$) parameter vectors for each LN layer are learned during training and are **fixed** during inference. Crucially, these parameters are shared across all token positions within a layer, meaning a single, layer-specific transformation is applied to every token in a sequence. This ensures that the geometric manifold each LN layer projects onto is also **fixed and constant across all time steps**. This fixed manifold acts as a powerful regularizer, but its role is best understood by distinguishing between two scenarios: the idealized, infinite-context Transformer and the practical, sliding-window Transformer.

**Stability in the Idealized Transformer: Process Consistency**

In a standard decoder-only Transformer with an infinite context window, the causal attention mask ensures that the representation of a token at position `p` is a function only of the tokens at positions `1...p`. Consequently, the representations of previously generated tokens are immutable; they are computed once and do not change as new tokens are appended. This immutability is the foundation of the KV Cache optimization used in modern inference engines.

In this idealized setting, the stability provided by the fixed manifold is not about correcting the "drift" of a single token's representation over time. Instead, it ensures **process consistency**. At any given time step `t`, the model processes the entire sequence of `t-1` tokens in parallel. The fixed manifold at layer `l`, $M_l$, guarantees that the vectors for `tok_1`, `tok_2`, ..., and `tok_{t-1}` are all projected onto the same geometric surface before being processed by the MHA or FFN. This creates a stable and predictable computational regime, ensuring the "rules of the game" for processing any token, at any position, remain constant. Our framework provides a novel, geometric explanation for this architectural stability, moving beyond purely statistical or empirical observations.

**Stability in the Practical Transformer: Taming Representational Drift**

In practice, to handle sequences longer than their training context length (e.g., 4096 tokens), models are often deployed with a **sliding context window**. In this scenario, as new tokens are generated, the oldest tokens are evicted from the context. This means the causal history for every token within the window changes at every step, forcing a re-computation of their representations.

Here, **representational drift is a real and significant phenomenon**. The vector for a token that is at position `p` in the window at time `t` will be different from its representation at time `t+1` when it has shifted to position `p-1`. In this challenging but common scenario, the role of the fixed manifold as a **geometric stabilizer** becomes even more critical. The repeated re-projection of these constantly changing token vectors onto the same fixed manifold at each layer and each time step acts as a powerful regularizer. It prevents the model's internal state from spiraling into chaotic or exploding regions of the embedding space, thereby ensuring coherence over extremely long generations.

In both the idealized and practical settings, the fixed manifold of Layer Normalization is a cornerstone of the Transformer's dynamic stability, providing a robust geometric foundation for coherent, long-form generation. This two-part explanation, distinguishing between the idealized and practical cases, offers a more nuanced understanding of stability than is typically discussed, grounding it in the specific geometric constraints of the architecture.

## 4 The Information Filter: A New Role for the FFN

With a stable geometric canvas provided by LayerNorm, we can now reinterpret the role of the Feed-Forward Network (FFN). We argue the FFN is not a generic function approximator but a specialized, **manifold-based information filter**, whose power derives from the Universal Approximation Theorem, which states that a two-layer network with a non-linear activation can approximate any continuous function, given sufficient width.

### 4.1 The FFN as a Manifold Processor

The FFN sub-layer computes `FFN(y)`, where `y` is a point on the input manifold `M_in`. The FFN itself consists of two linear layers separated by a non-linearity (e.g., GELU).

1. **Expansion (`h = GELU(y*W_1 + b_1)`):** The first linear layer, `W_1`, expands the dimensionality from `d_model` to `d_ffn` (typically `4 * d_model`). Geometrically, this is a learned **orientation module**. It takes the input manifold `M_in` and performs a linear transformation (rotation, scaling, translation) to place it in a specific orientation within the higher-dimensional `d_ffn` space. This new manifold is `M_h`.
2. **Contraction (`o = h*W_2 + b_2`):** The second linear layer, `W_2`, projects the result back down to `d_model`. Geometrically, this is a **summarization module** that learns to interpret the result of the intermediate processing step.

The critical question is: what is the purpose of the non-linear activation between these two steps?

### 4.2 The Activation as a Complexity-Reducing Deformation

The conventional answer, "to provide non-linearity," is true but insufficient. It does not explain why a simple, information-destroying function like ReLU (`max(0, x)`) or GELU (which approximates ReLU) is so effective.

We propose an information-theoretic interpretation: the activation function acts as a **fixed geometric filter** that performs a **complexity-reducing deformation** on the manifold `M_h`.

- **Mechanism:** Functions like ReLU and GELU are non-linear, but they have a specific character: they map a large portion of their input domain (e.g., all negative numbers) to a single, simple output (zero). This is an act of **lossy compression**.
- **Geometric Interpretation:** The activation function acts as a fixed stencil on the `d_ffn` space. The `W1` orientation module learns to position the manifold `M_h` such that the coordinates corresponding to **irrelevant or noisy features** for a given token fall into the filter's "discard" region (i.e., the negative half-spaces). The activation function then geometrically collapses these regions of the manifold towards the origin, effectively erasing that information.
- **Information-Theoretic Consequence:** This deformation is a direct, mechanistic implementation of **Kolmogorov complexity reduction**. A vector with many zeros is algorithmically simpler to describe than a dense vector. The FFN learns to simplify its representation by strategically destroying information.

### 4.3 The FFN as a Learned, Context-Dependent Feature Selector

This synthesis provides a powerful, purposeful model for the FFN's function. The FFN is a two-stage feature selection mechanism:

1. The **learned orientation module** (`W_1`) intelligently positions the token's representation relative to the filter.
2. The **fixed geometric filter (the activation)** performs a complexity-reducing deformation, discarding the features that `W_1` has positioned in the "off" region.

The FFN's role is to compute a filtered update vector. The residual connection then adds this intelligently simplified update to the original token representation. This allows the model to refine a token's meaning based only on the essential features that survived the filtering process.

## 5 The Anisotropic Processor: A Geometric View of Multi-Head Attention

Having established Layer Normalization as a "Geometric Stabilizer" and the FFN as a "Manifold-based Information Filter," we now turn our attention to the Multi-Head Attention (MHA) sub-block. The conventional view of MHA focuses on its ability to weigh the importance of different tokens. We propose a complementary geometric interpretation that frames MHA as a structured, **anisotropic processor**. This view reveals a deep design principle in the Transformer block: a repeating rhythm of isotropic regularization followed by anisotropic specialization, a `symmetrize-and-structure` cycle that appears fundamental to its ability to process information.

### 5.1 The Isotropic-Anisotropic Dichotomy: A Symmetrize-and-Structure Principle

The Transformer block exhibits a remarkable architectural pattern. Every major computational unit (MHA, FFN) is preceded by an operation (LayerNorm) with a fundamentally opposite geometric character.

1. **Symmetrize (Isotropic Projection):** As established in Section 2, LayerNorm is an **isotropic** operator. It treats all `d_model` dimensions of a token vector identically, projecting the vector onto a symmetric, hyperspherical manifold. This act of regularization creates a clean, standardized, and symmetric "canvas" by factoring out noisy variations in magnitude and placing all representations on an equal geometric footing.
2. **Structure (Anisotropic Processing):** The MHA block, which acts upon this clean canvas, is a fundamentally **anisotropic** operator. As we will detail, its core mechanism intentionally breaks the symmetry of the `d_model` space, imposing a specific, learned structure to enable specialized computation.

This `symmetrize-and-structure` sequence is a powerful design principle. The isotropic projection provides stability and a common geometric ground, while the subsequent anisotropic processing allows for a sophisticated, "divide and conquer" approach to computation.

### 5.2 The Geometry of Anisotropy: Axis-Aligned Subspace Decomposition

The anisotropy of MHA is not random; it is a highly structured feature that originates from the `reshape` operation used to create the multiple heads. Before the attention calculation, the `d_model`-dimensional representation of each token is reshaped into `H` separate head vectors, each of dimension `d_v` (where $d_{model} = H \times d_v$). This is a hard, **axis-aligned partitioning** of the vector space. For example, axes `0` to `d_v-1` are grouped into Head 1, axes `d_v` to `2d_v-1` into Head 2, and so on.

A profound implication of this design is that the network is **strongly incentivized to learn to encode meaning in its axis indices**. For the partitioning to be useful, the optimization process must discover a consistent "data layout," placing semantically related features onto contiguous axes that will be processed together by a dedicated attention head.

This architectural choice provides an elegant solution to a core challenge of operating in high-dimensional spaces. In a space with thousands of dimensions, two randomly chosen vectors are almost always nearly orthogonal. This "curse of orthogonality" would make it difficult for a single, global dot-product attention mechanism to learn meaningful similarity scores. The MHA's anisotropic partitioning circumvents this problem. By decomposing the high-dimensional space into a set of lower-dimensional subspaces (the heads), it creates "pockets" of non-trivial geometry where dense, meaningful, non-orthogonal relationships can be learned effectively. The anisotropy can therefore be seen as a **highly effective strategy** for imposing a learnable, interactive structure on an otherwise uniform and non-interactive high-dimensional space.

### 5.3 Attention as Relational Computation in Specialized Bases

Within each of these anisotropically defined subspaces, the MHA block performs another layer of specialization. The projection of the input vectors into Query (Q), Key (K), and Value (V) matrices can be interpreted as a learned **change of basis**.

Each head does not compute attention in the general-purpose basis of the main residual stream. Instead, it learns three specialized linear projections ($W^Q, W^K, W^V$) that transform the input into a new coordinate system optimized for its specific task. This allows different heads to specialize in detecting different kinds of inter-token relationships:

- **Head 1** might learn a basis optimized for resolving syntactic dependencies, where the dot product between Q and K vectors effectively measures grammatical agreement.
- **Head 2** might learn a basis optimized for tracking semantic similarity or opposition.
- **Head 3** might learn a basis optimized for anaphora resolution, where the Q vector for a pronoun like "it" is projected to be highly similar to the K vector of its antecedent.

This view frames the MHA block as performing a two-level decomposition: first, an anisotropic partitioning of the feature space into subspaces, and second, a further projection within each subspace into specialized computational bases. This allows the model to analyze the input sequence from multiple, parallel, and specialized relational perspectives simultaneously.

### 5.4 Future Research Directions

This geometric interpretation of MHA as an anisotropic processor opens up several new and promising avenues for research.

**Probing for Axis-Encoded Meaning:** The central hypothesis that the network learns to place semantically related features on contiguous axes is empirically testable. One could analyze the activation patterns of different heads when processing specific linguistic phenomena. A more direct test would involve targeted ablation studies: measure the performance degradation when shuffling axes *within* a single head's designated block versus shuffling the same number of axes *across* different head-blocks. Our framework would predict a significantly larger performance drop in the latter case, which would provide strong evidence for learned, axis-dependent feature grouping.

**Generalizing the Anisotropy with "Soft" Heads:** The current `reshape` operation creates a hard, block-diagonal partitioning. We can model this partitioning function as a periodic "impulse train," where an axis belongs 100% to one head and 0% to all others. This framing invites a natural generalization: what if we used other periodic functions to define the heads? For example, using overlapping sine or triangular waves could create "soft" heads, where an axis might contribute 80% of its activation to Head `h` and 20% to Head `h+1`. This would allow for more flexible and

potentially more powerful feature grouping, moving from a discrete to a continuous model of subspace decomposition.

**Connecting Head Specialization to Manifold Geometry:** An open theoretical question is how the anisotropic processing of MHA interacts with the geometric manifold created by the preceding LayerNorm. Does the MHA block as a whole learn to operate on the global geometry of the manifold? Or do individual heads learn to specialize on different regions or curvatures of the input manifold, with the axis-aligned partitioning serving as a routing mechanism to direct different parts of the manifold to the appropriate specialist head? Answering this would provide a truly unified geometric theory of the entire Transformer block.

### 5.5 The Manifold and the Grid: A Productive Incommensurability

Our framework reveals a final, subtle tension at the heart of the `symmetrize-and-structure` cycle. The `symmetrize` step (LayerNorm) projects token representations onto a smooth, (`d_model-2`)-dimensional manifold. The `structure` step (MHA) then processes this manifold using a rigid, `d_model`-dimensional grid, partitioned into `H` discrete heads. There is a fundamental **incommensurability** between the dimensionality of the data and the dimensionality of the processor.

We hypothesize that this mismatch is not a design flaw but a powerful form of implicit regularization. It is mathematically impossible to perfectly and evenly partition a (`d_model-2`)-dimensional object into subspaces defined on a `d_model`-dimensional grid. This geometric friction prevents the model from learning a brittle, "too-perfect" alignment between its learned features and the arbitrary boundaries of the attention heads. Instead, the model is forced to learn more **distributed and robust representations** that are resilient to the slight geometric aliasing that occurs at the LN/MHA interface. This productive tension between the continuous geometry of the manifold and the discrete architecture of the grid may be a key, unappreciated reason for the robustness and generalizability of the Transformer architecture.

## 6 Discussion and Future Work

**A Unified View: The Symmetrize-and-Structure Principle:** The geometric and information-theoretic framework presented here unifies the roles of LayerNorm, Multi-Head Attention, and the FFN into a single, coherent narrative. The Transformer block is not an arbitrary collection of layers but a sophisticated computational engine operating on a `symmetrize-and-structure` principle. The process begins with an isotropic projection by LayerNorm, which acts as a **Geometric Stabilizer** to create a stable, symmetric manifold. This is followed by the **Anisotropic Processor** of MHA, which imposes a learned, axis-aligned structure on this manifold to enable specialized, parallel computation. Finally, after another stabilization step, the **Information Filter** of the FFN performs a complexity-reducing deformation to perform context-dependent feature selection.

**Consilience with the Renormalization Group:** This perspective enriches other theoretical models, such as the view of deep networks as performing a Renormalization Group (RG) flow. In that analogy, the layers correspond to a change in scale. Our framework provides the concrete mechanism for this flow: the `symmetrize-and-structure` cycle is a robust implementation of an RG step. The isotropic projections (`LayerNorm`) act as the coarse-graining or regularization operators that stabilize the flow, while the anisotropic processors (`MHA`, `FFN`) act as the update-computation engines that calculate the new effective couplings for the next scale. This suggests the Transformer has convergently discovered a fundamental principle of information processing in complex, multi-scale systems.

**A Generative Framework for Future Work:** This framework also opens several new avenues for empirical research and model design. Because the core geometric claims of our framework are derived mathematically from the architecture's definitions, the primary value of empirical testing is not to "validate" these truths, but to use them as a lens to probe the model's learned behavior. Such experiments are likely to yield serendipitous inspirations and reveal further puzzles about the nature of the learned representations.

**Probing the Latent Geometry:** As detailed in Section 4.4, the geometric view of MHA inspires concrete proposals for probing axis-encoded meaning and designing novel "soft head" architectures. Beyond MHA-specific inquiries, our framework's most direct global prediction is that Layer Normalization reduces the effective dimensionality of the latent space by two. This can be empirically verified by computing the covariance matrix of post-LN activation vectors; our theory predicts this matrix will have **two** eigenvalues that are orders of magnitude smaller than the others, corresponding to the zero-mean and constant-norm constraints.

**Designing New Components:** A deeper geometric understanding can also inspire the design of entirely new normalization or activation layers based on explicit geometric principles. Furthermore, this framework provides a new vocabulary for failure analysis, allowing us to ask if model errors can be understood as instances where representations fall "off" the learned manifolds or are improperly processed by the anisotropic components.

**A Final, Speculative Projection: The Evolution of Structure Itself:** Our framework's reinterpretation of MHA's fixed partitioning as a specific "impulse train"-like function naturally invites a generalization to "soft heads," where this rigid structure is replaced by a learnable, parameterized function. This step alone, moving a core architectural heuristic from a static rule to a dynamic parameter, points toward a more profound future. It is conceivable that in a continuous, AI-supervised training regime, the partitioning heuristic itself could become an object of evolution, with a "supervisor" AI dynamically modifying the "worker" AI's internal structure to best suit the data domain it currently faces. This vision culminates in the proposal for a fully trainable, anisotropic LayerNorm, as detailed in the $\omega + 1$ addendum. Such a component, which would learn the `symmetrize` operation through a parallel, iterative process, would resolve the final architectural tension of the Transformer block, making the `symmetrize-and-structure` cycle fully learnable, architecturally consistent, and more biologically plausible. This points towards a future where our models do not just learn *within* a fixed architecture, but learn *how to architect themselves* for optimal information processing.

## 7 Conclusion

We have proposed a new, unified framework for understanding the Layer Normalization, Multi-Head Attention, and Feed-Forward Network components of the Transformer architecture. We argue that their roles are not merely statistical or algebraic, but are fundamentally geometric and information-theoretic, operating in a `symmetrize-and-structure` cycle.

LayerNorm acts as a **Geometric Stabilizer**, performing an isotropic projection of representations onto fixed, layer-specific manifolds to ensure dynamic stability. Multi-Head Attention then acts as an **Anisotropic Processor**, imposing a learned, axis-aligned structure on this symmetric space to enable specialized, parallel computation. Finally, the FFN acts as a **Manifold-based Information Filter**, using a complexity-reducing deformation to perform context-dependent feature selection.

This unified perspective provides a more satisfying, first-principles explanation for the Transformer's design and effectiveness. It reveals a deep architectural logic that balances symmetry with structure,

and stability with specialization. This framework offers a powerful new lens for future research into the mechanisms of deep learning, suggesting new ways to probe, analyze, and design the next generation of intelligent systems.

## 8 References


Ba, J. L., Kiros, J. R., & Hinton, G. E. (2016). Layer normalization. *arXiv preprint arXiv:1607.06450*.

He, K., Zhang, X., Ren, S., & Sun, J. (2016). Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition* (pp. 770-778).

Hendrycks, D., & Gimpel, K. (2016). Gaussian Error Linear Units (GELUs). *arXiv preprint arXiv:1606.08415*.

Mehta, P., & Schwab, D. J. (2014). An exact mapping between the Variational Renormalization Group and Deep Learning. *arXiv preprint arXiv:1410.3831*.

Nair, V., & Hinton, G. E. (2010). Rectified linear units improve restricted boltzmann machines. In *Proceedings of the 27th international conference on machine learning (ICML-10)* (pp. 807-814).

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., ... & Polosukhin, I. (2017). Attention is all you need. *Advances in neural information processing systems, 30*.

Xiong, R., Yang, Y., He, D., Zheng, S., Zheng, S., Xing, C., ... & Liu, T. Y. (2020). On Layer Normalization in the Transformer Architecture. In *Proceedings of the 37th International Conference on Machine Learning, PMLR 119* (pp. 10525-10535).


## 9 $\omega + 1$: And One More Thing: a Trainable LayerNorm?

Our analysis has revealed the Transformer block to be a sophisticated engine operating on a `symmetrize-and-structure` principle. We have identified Layer Normalization as the **Geometric Stabilizer**, an isotropic operator that projects representations onto a fixed manifold, thereby creating a symmetric canvas for computation. This is followed by the **Anisotropic Processor** of Multi-Head Attention, which imposes a learned, structured partitioning on this canvas. This `isotropic -> anisotropic` cycle is a deep and powerful design pattern.

Yet, it presents a final, subtle tension. The `symmetrize` step is performed by a fixed, analytical, and global algorithm, making it a rigid outlier in an otherwise learnable, parallel, and structured system. This invites a final, speculative question: Now that we understand the *function* of Layer Normalization, can we design a better *implementation*—one that is more harmonious with the architecture it serves and more congruent with the principles of neural computation?

We propose that the next frontier is to replace the standard LayerNorm algorithm with a learnable, parallel, and iterative block. This "symmetrizing block" would be a neural network module in its own right, designed to perform the same geometric function of projecting a vector onto a stable manifold, but through a different mechanism. Crucially, this block would be designed to be **anisotropic from the start**. It would be composed of `H` parallel heads, each responsible for a subset of the `d_model` dimensions, mirroring the structure of the MHA block it precedes. This would resolve the architectural tension, transforming the `isotropic -> anisotropic` cycle into a more elegant and consistent `anisotropic symmetrize -> anisotropic structure` rhythm.

Such a proposal is motivated not only by the pursuit of architectural elegance but also by biological plausibility. It is difficult to imagine a biological system implementing a global "compute mean and

variance" operation. A parallel, iterative algorithm, where local computational units communicate to converge on a global property, is a far more likely model for how a brain might achieve a similar normalization function.

The engineering challenges would be significant. A randomly initialized block would likely re-introduce the very instabilities that Pre-LN architectures solve, necessitating a careful pre-training regimen (perhaps by first learning to mimic the standard LN function) and a slow learning rate schedule. However, the parameters of this block could be learned once and shared universally, not just across all layers of a single model, but potentially as a universal, pre-trained "symmetrizing chip" for all Transformers.

This is not a finished idea, but a research direction opened up by the geometric framework. It is an invitation to move beyond analyzing the architectures we have and begin designing new components based on the principles we have uncovered. It is a step towards a future where the fundamental building blocks of our models are not just effective, but are also learnable, principled, and perhaps, a little closer to the way intelligence is organized in the natural world.

## 10 Addendum: DecodeBlock Pseudocode

This addendum details the step-by-step computations within a single `DecodeBlock` of a decoder-only Transformer. We use TeX notation and explicitly state the dimensions (rows $\times$ columns) of all data and weight matrices.

### 10.1 Notation and Dimensions

Let the primary hyperparameters of the model be:

- $B$: Batch size (number of sequences processed in parallel).
- $T$: Sequence length (number of tokens in each sequence).
- $d_{\text{model}}$: The main embedding dimension of the model.
- $H$: The number of parallel attention heads.
- $d_k$: The dimension of the Query and Key vectors for each head.
- $d_v$: The dimension of the Value vector for each head.

For standard Transformer architectures, the dimensions are constrained such that the total capacity across all heads equals the model's main dimension: $H \cdot d_k = H \cdot d_v = d_{\text{model}}$.

### 10.2 The DecodeBlock

A `DecodeBlock` transforms an input sequence representation into an output sequence representation of the same size, following the pre-normalization variant common in modern GPT-style models.

- **Input to Block**: $X^{(L-1)} \in \mathbb{R}^{B \times T \times d_{\text{model}}}$. This is a batch of $B$ sequences, each with $T$ token vectors of dimension $d_{\text{model}}$.
- **Output of Block**: $X^{(L)} \in \mathbb{R}^{B \times T \times d_{\text{model}}}$.

The computation within the block proceeds as follows:

1. $X_{\text{norm1}} = \text{LayerNorm1}(X^{(L-1)})$
2. $Y = X^{(L-1)} + \text{MHA}(X_{\text{norm1}})$
3. $Y_{\text{norm2}} = \text{LayerNorm2}(Y)$
4. $X^{(L)} = Y + \text{FFN}(Y_{\text{norm2}})$

We now detail the computations within the Multi-Head Attention (MHA) block.

### 10.3 Detailed Computation within the Multi-Head Attention (MHA) Block

The MHA block takes the normalized input $X_{\text{norm1}}$ and produces an output of the same dimension.

- **Input to MHA**: $X_{\text{in}} \in \mathbb{R}^{B \times T \times d_{\text{model}}}$ (where $X_{\text{in}} = X_{\text{norm1}}$).
- **Output of MHA**: $\text{MHA}_{\text{out}} \in \mathbb{R}^{B \times T \times d_{\text{model}}}$.

### 10.4 MHA Sub-block 2: Learned Projections for Q, K, and V

The first step is to project the input $X_{\text{in}}$ into three separate matrices: Query (Q), Key (K), and Value (V). This is done using three distinct, learned linear projection weight matrices.

- **Weight Matrices**:
  - $W^Q \in \mathbb{R}^{d_{\text{model}} \times d_{\text{model}}}$ (rows $\times$ columns: $d_{\text{model}} \times d_{\text{model}}$)
  - $W^K \in \mathbb{R}^{d_{\text{model}} \times d_{\text{model}}}$ (rows $\times$ columns: $d_{\text{model}} \times d_{\text{model}}$)
  - $W^V \in \mathbb{R}^{d_{\text{model}} \times d_{\text{model}}}$ (rows $\times$ columns: $d_{\text{model}} \times d_{\text{model}}$)
- **Computation**:

$$Q' = X_{\text{in}} W^Q \quad | \quad K' = X_{\text{in}} W^K \quad | \quad V' = X_{\text{in}} W^V$$

- **Resulting Dimensions**: The resulting matrices $Q'$, $K'$, and $V'$ each have the dimension $\mathbb{R}^{B \times T \times d_{\text{model}}}$.
- **Prepare for Multi-Head Processing**: These matrices are then reshaped and transposed to split the $d_{\text{model}}$ dimension across the $H$ attention heads.
  - $Q = \text{reshape}(Q') \to \mathbb{R}^{B \times T \times H \times d_k} \xrightarrow{\text{transpose}} \mathbb{R}^{B \times H \times T \times d_k}$
  - $K = \text{reshape}(K') \to \mathbb{R}^{B \times T \times H \times d_k} \xrightarrow{\text{transpose}} \mathbb{R}^{B \times H \times T \times d_k}$
  - $V = \text{reshape}(V') \to \mathbb{R}^{B \times T \times H \times d_v} \xrightarrow{\text{transpose}} \mathbb{R}^{B \times H \times T \times d_v}$

### 10.5 MHA Sub-block 1: Scaled Dot-Product Attention

This core operation is performed independently for each head in parallel.

- **Inputs**: $Q, K \in \mathbb{R}^{B \times H \times T \times d_k}$ and $V \in \mathbb{R}^{B \times H \times T \times d_v}$.
- **Step 1: Compute Attention Scores**: The dot product between every query vector and every key vector is computed.

$$\text{Scores} = Q \cdot K^T$$

  The matrix multiplication is performed on the last two dimensions. The dimensions are: $(\mathbb{R}^{B \times H \times T \times d_k}) \cdot (\mathbb{R}^{B \times H \times d_k \times T}) \to \mathbb{R}^{B \times H \times T \times T}$. The resulting `Scores` matrix contains, for each head, a $T \times T$ matrix of similarity scores between all pairs of tokens in the sequence.
- **Step 2: Scale, Mask, and Softmax**: The scores are scaled to stabilize gradients, masked to enforce causality (i.e., a token cannot attend to future tokens), and normalized into a probability distribution using softmax.

$$\text{AttentionWeights} = \text{softmax}\left(\frac{\text{Scores}}{\sqrt{d_k}} + M\right)$$

- The mask $M \in \mathbb{R}^{T \times T}$ is a matrix where $M_{ij} = -\infty$ for $j > i$ (upper triangle) and $M_{ij} = 0$ otherwise. This is broadcast across the batch and head dimensions.
- The softmax is applied along the last dimension (i.e., row-wise for each $T \times T$ matrix).
- The resulting `AttentionWeights` matrix has dimension $\mathbb{R}^{B \times H \times T \times T}$.

- **Step 3: Compute Head Outputs**: The attention weights are used to compute a weighted sum of the value vectors.

$$\text{Heads}_{\text{out}} = \text{AttentionWeights} \cdot V$$

The dimensions are: $(\mathbb{R}^{B \times H \times T \times T}) \cdot (\mathbb{R}^{B \times H \times T \times d_v}) \to \mathbb{R}^{B \times H \times T \times d_v}$.

### 10.6 MHA Sub-block 3: Concatenation of Heads

The outputs from all $H$ heads are combined back into a single tensor.

- **Input**: $\text{Heads}_{\text{out}} \in \mathbb{R}^{B \times H \times T \times d_v}$.
- **Computation**: The operation is a transpose followed by a reshape, which is equivalent to concatenation.
  1. $\text{Heads}_{\text{out}} \xrightarrow{\text{transpose}} \mathbb{R}^{B \times T \times H \times d_v}$
  2. $\xrightarrow{\text{reshape}} \mathbb{R}^{B \times T \times (H \cdot d_v)}$
- **Resulting Matrix**: Since $H \cdot d_v = d_{\text{model}}$, the final concatenated matrix is:

$$\text{Concatenated} \in \mathbb{R}^{B \times T \times d_{\text{model}}}$$

### 10.7 MHA Sub-block 4: Final Linear Projection

The concatenated output is passed through a final linear layer to produce the MHA block's final output.

- **Input**: $\text{Concatenated} \in \mathbb{R}^{B \times T \times d_{\text{model}}}$.
- **Weight Matrix**:
  - $W^O \in \mathbb{R}^{d_{\text{model}} \times d_{\text{model}}}$ (rows $\times$ columns: $d_{\text{model}} \times d_{\text{model}}$)
- **Computation**:

$$\text{MHA}_{\text{out}} = \text{Concatenated} \cdot W^O$$

- **Final MHA Output**: The final output of the MHA block has dimension $\mathbb{R}^{B \times T \times d_{\text{model}}}$, matching the dimension of the block's input. This output is then passed to the residual connection and the subsequent FFN sub-layer.

## 11 Addendum: A Geometric Analogy for Implicit Non-Linearity

### 11.1 The Core Idea

The core idea this example illustrates is that when a set of points is confined to a lower-dimensional, curved surface (a manifold), a simple linear operation in the higher-dimensional "ambient" space does not act linearly on the surface itself. To an observer living on the surface, the transformation appears non-linear and complex.

This example provides an intuitive analogy for the computations within a Transformer block. The 3D ambient space corresponds to the Transformer's high-dimensional embedding space ($\mathbb{R}^{d_{\text{model}}}$).

The 2D spherical surface represents the curved, lower-dimensional manifold onto which Layer Normalization projects token vectors. Finally, the simple 3D linear transformation is analogous to the linear weight matrices applied by the MHA and FFN layers.

### 11.2 A 3D Example: Shearing a Sphere

Let's set up our example.

**1. The Setup: A 2D Surface in 3D Space**

Imagine our set of points are not just any points in 3D space, but are all constrained to lie on the surface of a sphere of radius `r` centered at the origin.

- **The Manifold:** This spherical surface is our 2D manifold. Although any point `p` on it has three coordinates `(x, y, z)`, they are not independent.
- **The Constraint:** The coordinates must satisfy the equation:

$$x^2 + y^2 + z^2 = r^2$$

  This constraint reduces the degrees of freedom from 3 to 2. This is analogous to how LayerNorm constrains the `d_model` dimensions of a token vector.

**2. The Transformation: A Linear Operation in 3D**

Now, let's define a simple **linear transformation** that we will apply to every point in our 3D space. We'll use a "shear" transformation, which shifts points in one direction by an amount proportional to their coordinate in another direction.

Let's define a shear along the x-axis that is proportional to a point's z-coordinate. The transformation matrix `W` and its effect on a point `p = (x, y, z)` is:

$$W = \begin{pmatrix} 1 & 0 & s \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

$$p' = Wp = \begin{pmatrix} 1 & 0 & s \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ z \end{pmatrix} = \begin{pmatrix} x + sz \\ y \\ z \end{pmatrix}$$

This is a perfectly linear transformation. It simply adds `s` times the z-coordinate to the x-coordinate. If we apply this to a cube, it would tilt into a slanted parallelepiped.

**3. The Result: A Non-Linear Transformation on the 2D Surface**

What happens when we apply this linear shear to every point on our sphere? The sphere is distorted into a new shape—a "sheared sphere," which is a type of ellipsoid.

The crucial question is: how can we describe the transformation that happened *on the surface*? A great way to see the non-linearity is to look at the "shadow" the sphere casts on the xy-plane before and after the transformation.

- **Before Transformation:** The projection of the original sphere onto the xy-plane is a filled disk of radius `r`. A point in this disk is `(x, y)`.
- **After Transformation:** A point `(x, y, z)` on the original sphere is moved to `(x + sz, y, z)`. The projection of this new point onto the xy-plane is `(x', y') = (x + sz, y)`.

Now, let's define the transformation purely in terms of the 2D coordinates `(x, y)` of the projection. To do this, we must substitute for `z`. From our sphere's constraint equation, we know:

$$z = \pm\sqrt{r^2 - x^2 - y^2}$$

Substituting this into our transformation for the projected points gives us:

$$(x, y) \to (x', y') = \left(x \pm s\sqrt{r^2 - x^2 - y^2}, \quad y\right)$$

This is the rule that maps a point from the original circular shadow to its corresponding point in the new, distorted shadow.

**This transformation is clearly non-linear.** It involves square roots and squared terms. A simple, clean, linear shear in 3D has induced a complex, non-linear "warping" in the 2D projection space.

### 11.3 Conclusion and Connection to Transformers

This example provides an intuitive analogy for what happens inside a Transformer block.

1. **LayerNorm** takes the token vectors, which could be anywhere in `d_model`-dimensional space, and projects them onto a specific, curved (`d_model-2`)-dimensional manifold (analogous to our sphere).
2. The **MHA and FFN layers** then apply linear transformations (weight matrices, analogous to our shear matrix `W`) to these constrained vectors.
3. From the "extrinsic" view of the full `d_model` space, this operation is linear. But from the "intrinsic" view of the lower-dimensional manifold where the token representations actually live, the transformation is powerfully **non-linear**.

This interplay—projecting onto a curved surface and then applying a linear map—is a fundamental source of the Transformer's computational expressiveness. It's a more subtle and geometric form of non-linearity than that provided by explicit activation functions like GELU, and it happens at every sub-layer of the network.