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

Anonymous Author(s)

Affiliation

Address

email

Abstract

1 The Transformer architecture has become the foundation of modern artificial intelligence, yet the first-principles reasoning for its design remains surprisingly shallow. 2 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 3 components. This paper proposes a new, unified framework that reframes these components through a geometric and information-theoretic lens. We argue that the 4 Transformer block operates on a **symmetrize-and-structure** principle. The process begins with LN acting as a **Geometric Stabilizer**, an isotropic operator that 5 projects token representations onto a fixed, $(d_{\text{model}}-2)$ -dimensional manifold, ensuring dynamic stability. This symmetric “canvas” is then processed by MHA, 6 which we identify as an **Anisotropic Processor** that performs **Axis-Aligned Subspace Decomposition**. A profound implication of this design is that the network is 7 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** 8 **Filter** that performs a complexity-reducing deformation to select for relevant features. This unified geometric perspective provides a more powerful, first-principles 9 understanding of the Transformer’s effectiveness and stability. It culminates in a proposal for a fully trainable, anisotropic LayerNorm, which would make the 10 **symmetrize-and-structure** cycle architecturally consistent and biologically plausible, opening a new frontier in model design.

22 1 Introduction

23 The Transformer architecture, introduced by Vaswani et al. (2017), has revolutionized natural language 24 processing and beyond. Its success is largely attributed to the self-attention mechanism, which allows 25 for the modeling of global dependencies in a highly parallelizable manner. However, the architecture’s 26 remarkable stability and performance also rely on two other ubiquitous components in every block: a 27 Layer Normalization (LN) layer and a position-wise Feed-Forward Network (FFN).

28 The conventional understanding of these components is largely functional and historical. Layer 29 Normalization (Ba et al., 2016) is typically justified as a technique to stabilize training by normalizing 30 the distribution of activations, thereby reducing “internal covariate shift.” The FFN is explained as 31 being necessary to provide non-linear expressive power, as a stack of linear attention layers would 32 otherwise collapse into a single, less powerful linear transformation.

33 While these explanations are not incorrect, they are unsatisfying. They do not explain, for instance, 34 the precise mathematical synergy between LayerNorm and the $\sqrt{d_k}$ scaling factor in attention. 35 Nor do they provide a compelling rationale for why the specific form of the FFN—an expansion, a 36 ReLU/GELU activation, and a contraction—is so effective. They describe *what* the components do at

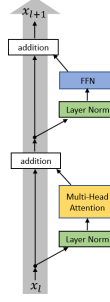


Figure 1: pre-LN Transformer Block

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

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

73 3 The Geometric Stabilizer: A New Role for LayerNorm

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

77 3.1 The Geometry of Normalization: From $\mathbb{R}^{d_{\text{model}}}$ to a $(d_{\text{model}} - 2)$ -Manifold

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

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

80 where μ and σ^2 are the mean and variance of the components of \mathbf{x} . This operation is not just a
81 statistical rescaling; it is a geometric projection. The output vector \mathbf{z} is not free to point anywhere in
82 $\mathbb{R}^{d_{\text{model}}}$. It is constrained to lie on a specific, lower-dimensional surface defined by two properties:

- 83 1. **Zero Mean:** The sum of its components is zero: $\sum_{i=1}^{d_{\text{model}}} z_i = 0$. This equation defines a
84 hyperplane in $\mathbb{R}^{d_{\text{model}}}$ that passes through the origin. This single linear constraint reduces the
85 degrees of freedom by one.
- 86 2. **Unit Variance / Constant Norm:** The sum of the squares of its components is constant:
87 $\sum_{i=1}^{d_{\text{model}}} z_i^2 = d_{\text{model}} \frac{\sigma^2}{\sigma^2 + \epsilon} \approx d_{\text{model}}$. This equation confines the vector \mathbf{z} to the surface of a
88 hypersphere of radius $\sqrt{d_{\text{model}}}$.

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

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

97 3.2 Synergy with Attention: Aligning Representational Space with Computation

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

- 100 1. **Factoring out Magnitude:** By forcing all token vectors onto a bounded manifold, LN
101 effectively factors out the vector’s magnitude as a primary information carrier. A vector’s
102 “meaning” is no longer encoded in its length but in its **direction**—its specific location on the
103 manifold.
- 104 2. **Congruence with Attention:** The core operation of attention is the dot product, score
105 $= \mathbf{q} \cdot \mathbf{k}$. We know that $\mathbf{q} \cdot \mathbf{k} = \|\mathbf{q}\| \|\mathbf{k}\| \cos(\text{theta})$. Since the vectors used to
106 generate \mathbf{q} and \mathbf{k} originate from the LN manifold, their norms are constrained. Consequently,
107 the attention score becomes primarily a function of $\cos(\text{theta})$, the cosine similarity.

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

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

122 For two unit vectors on a hypersphere, this geodesic distance is simply the angle, θ , between
123 them. This angle is directly and linearly recoverable from the dot product that attention computes:
124 $\theta = \arccos(\mathbf{q} \cdot \mathbf{k})$. This is a crucial insight. It means the model has discovered a geometry
125 where its fundamental algebraic operation (the dot product) is a direct, uncorrupted measure of true
126 geometric distance. The “curse of dimensionality” is sidestepped by adopting a geometry where
127 angular separation *is* distance, perfectly aligning the representational space with the computational
128 mechanism.

129 3.3 The Fixed Manifold: A Mechanism for Dynamic Stability

130 The geometric view of Layer Normalization provides its most significant explanatory power when
131 we consider the dynamics of autoregressive generation. A key challenge for any generative model
132 is maintaining coherence and stability over long sequences. The “Geometric Stabilizer” framework
133 reveals a powerful mechanism by which the Transformer achieves this, though the precise nature of
134 the mechanism depends on the model’s operational context.

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

142 Stability in the Idealized Transformer: Process Consistency

143 In a standard decoder-only Transformer with an infinite context window, the causal attention mask
144 ensures that the representation of a token at position p is a function only of the tokens at positions
145 $1 \dots p$. Consequently, the representations of previously generated tokens are immutable; they are
146 computed once and do not change as new tokens are appended. This immutability is the foundation
147 of the KV Cache optimization used in modern inference engines.

148 In this idealized setting, the stability provided by the fixed manifold is not about correcting the “drift”
149 of a single token’s representation over time. Instead, it ensures **process consistency**. At any given
150 time step t , the model processes the entire sequence of $t-1$ tokens in parallel. The fixed manifold
151 at layer l , M_l , guarantees that the vectors for $\text{tok}_1, \text{tok}_2, \dots$, and $\text{tok}_{\{t-1\}}$ are all projected
152 onto the same geometric surface before being processed by the MHA or FFN. This creates a stable
153 and predictable computational regime, ensuring the “rules of the game” for processing any token,
154 at any position, remain constant. Our framework provides a novel, geometric explanation for this
155 architectural stability, moving beyond purely statistical or empirical observations.

156 Stability in the Practical Transformer: Taming Representational Drift

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

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

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

173 4 The Information Filter: A New Role for the FFN

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

179 4.1 The FFN as a Manifold Processor

180 The FFN sub-layer computes $\text{FFN}(y)$, where y is a point on the input manifold M_{in} . The FFN itself
 181 consists of two linear layers separated by a non-linearity (e.g., GELU).

- 182 1. **Expansion** ($h = \text{GELU}(y * W_1 + b_1)$): The first linear layer, W_1 , expands the dimen-
 183 sionality from d_{model} to d_{ffn} (typically $4 * d_{\text{model}}$). Geometrically, this is a learned
 184 **orientation module**. It takes the input manifold M_{in} and performs a linear transforma-
 185 tion (rotation, scaling, translation) to place it in a specific orientation within the higher-
 186 dimensional d_{ffn} space. This new manifold is M_h .
- 187 2. **Contraction** ($o = h * W_2 + b_2$): The second linear layer, W_2 , projects the result back
 188 down to d_{model} . Geometrically, this is a **summarization module** that learns to interpret
 189 the result of the intermediate processing step.

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

191 4.2 The Activation as a Complexity-Reducing Deformation

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

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

- 197 • **Mechanism:** Functions like ReLU and GELU are non-linear, but they have a specific
 198 character: they map a large portion of their input domain (e.g., all negative numbers) to a
 199 single, simple output (zero). This is an act of **lossy compression**.
- 200 • **Geometric Interpretation:** The activation function acts as a fixed stencil on the d_{ffn}
 201 space. The W_1 orientation module learns to position the manifold M_h such that the co-
 202 ordinates corresponding to **irrelevant or noisy features** for a given token fall into the
 203 filter’s “discard” region (i.e., the negative half-spaces). The activation function then geomet-
 204 rically collapses these regions of the manifold towards the origin, effectively erasing that
 205 information.
- 206 • **Information-Theoretic Consequence:** This deformation is a direct, mechanistic implemen-
 207 tation of **Kolmogorov complexity reduction**. A vector with many zeros is algorithmically
 208 simpler to describe than a dense vector. The FFN learns to simplify its representation by
 209 strategically destroying information.

210 4.3 The FFN as a Learned, Context-Dependent Feature Selector

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

- 213 1. The **learned orientation module** (W_1) intelligently positions the token’s representation
 214 relative to the filter.

215 2. The **fixed geometric filter (the activation)** performs a complexity-reducing deformation,
216 discarding the features that W_1 has positioned in the “off” region.

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

220 5 The Anisotropic Processor: A Geometric View of Multi-Head Attention

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

228 5.1 The Isotropic-Anisotropic Dichotomy: A Symmetrize-and-Structure Principle

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

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

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

244 5.2 The Geometry of Anisotropy: Axis-Aligned Subspace Decomposition

245 The anisotropy of MHA is not random; it is a highly structured feature that originates from the
246 **reshape** operation used to create the multiple heads. Before the attention calculation, the d_{model} -
247 dimensional representation of each token is reshaped into H separate head vectors, each of dimension
248 d_v (where $d_{\text{model}} = H \times d_v$). This is a hard, **axis-aligned partitioning** of the vector space. For
249 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.

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

254 This architectural choice provides an elegant solution to a core challenge of operating in high-
255 dimensional spaces. In a space with thousands of dimensions, two randomly chosen vectors are
256 almost always nearly orthogonal. This “curse of orthogonality” would make it difficult for a single,
257 global dot-product attention mechanism to learn meaningful similarity scores. The MHA’s anisotropic
258 partitioning circumvents this problem. By decomposing the high-dimensional space into a set of
259 lower-dimensional subspaces (the heads), it creates “pockets” of non-trivial geometry where dense,
260 meaningful, non-orthogonal relationships can be learned effectively. The anisotropy can therefore be
261 seen as a **highly effective strategy** for imposing a learnable, interactive structure on an otherwise
262 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.

6 Discussion and Future Work

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 creates a stable, symmetric manifold. MHA then imposes a learned, anisotropic structure on this manifold, partitioning the feature space to enable specialized, parallel computation. This is followed by another isotropic projection to re-stabilize the state, after which the FFN performs a final, complexity-reducing filtering operation.

313 This perspective enriches other theoretical models, such as the view of deep networks as performing
 314 a Renormalization Group (RG) flow. In that analogy, the layers correspond to a change in scale.
 315 Our framework provides the concrete mechanism for this flow: the `symmetrize-and-structure`
 316 cycle is a robust implementation of an RG step. The isotropic projections (LayerNorm) act as the
 317 coarse-graining or regularization operators that stabilize the flow, while the anisotropic processors
 318 (MHA, FFN) act as the update-computation engines that calculate the new effective couplings for the
 319 next scale.

320 This framework also opens several new avenues for empirical research and model design. Because
 321 the core geometric claims of our framework are derived mathematically from the architecture’s
 322 definitions, the primary value of empirical testing is not to “validate” these truths, but to use them
 323 as a lens to probe the model’s learned behavior. Such experiments are likely to yield serendipitous
 324 inspirations and reveal further puzzles about the nature of the learned representations.

325 **Probing the Latent Geometry:** Our framework makes several strong, testable predictions about the
 326 geometry of the latent space.

- 327 • **Dimensionality and Manifold Analysis:** The most direct prediction of our framework is
 328 that Layer Normalization reduces the effective dimensionality of the latent space by two.
 329 This can be empirically verified by collecting post-LN activation vectors for a large and
 330 diverse dataset and computing their covariance matrix. Our theory predicts that this matrix
 331 will have **two** eigenvalues that are orders of magnitude smaller than the others, corresponding
 332 to the zero-mean and constant-norm constraints. Furthermore, we can probe the learned
 333 gamma and beta parameters to analyze the specific geometry of the resulting manifolds
 334 at different layers. Do deeper layers learn “simpler” or “smoother” manifolds, as the RG
 335 analogy would predict?
- 336 • **Axis-Encoded Meaning:** The hypothesis that MHA’s anisotropy forces the network to
 337 encode meaning in axis indices is empirically testable. Targeted ablation studies that shuffle
 338 axes *within* a head’s designated block versus *across* different head-blocks could validate
 339 this claim. A significantly larger performance drop in the latter case would provide strong
 340 evidence for learned, axis-dependent feature grouping.
- 341 • **Head Specialization and Manifold Interaction:** An open question is how the anisotropic
 342 partitioning of MHA interacts with the global geometry of the input manifold. Do individual
 343 heads learn to operate on different regions or curvatures of the manifold, with the axis-
 344 aligned partitioning serving as a routing mechanism to direct different parts of the manifold
 345 to the appropriate specialist head?

346 **Designing New Components:** A deeper geometric understanding can inspire novel architectures.

- 347 • Could we design better normalization or activation layers by explicitly defining the target
 348 manifold’s geometry or the nature of the complexity-reducing filter?
- 349 • The “soft heads” proposal, which generalizes MHA’s hard partitioning to a continuous,
 350 overlapping one using different periodic functions, is a concrete example of a new component
 351 inspired by this framework.

352 **Understanding Failure Modes:** Can model failures be understood as instances where representations
 353 fall “off” the learned manifolds, are incorrectly routed by the anisotropic processor, or are improperly
 354 filtered by the FFN?

355 **A Final, Speculative Projection: The Evolution of Structure Itself:** Our framework’s re-
 356 interpretation of MHA’s axis-aligned partitioning as a fixed, “impulse train”-like function naturally
 357 invites a generalization to “soft heads,” where this rigid structure is replaced by a learnable, parame-
 358 terized function. This step alone, moving a core architectural heuristic from a static rule to a dynamic
 359 parameter, opens up a new design space for more flexible attention mechanisms.

360 Looking further into the future, this parameterization suggests an even more profound possibility.
 361 In a world of continuous, AI-supervised training, the partitioning heuristic itself could become an
 362 object of evolution. It is conceivable that a “supervisor” AI could learn to dynamically modify the
 363 partitioning function of a “worker” AI, adapting its internal information-processing strategy to best
 364 suit the data domain it currently faces—be it natural language, source code, or molecular structures.
 365 The function defining how the model decomposes problems would itself become a fluid, optimized
 366 component.

This speculative projection represents the ultimate expression of the *symmetrize-and-structure* principle. It 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. The static Transformer block, as we understand it today, may one day be seen not as an endpoint, but as a crucial step towards truly dynamic, self-organizing intelligent systems.

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 \rightarrow 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 \rightarrow anisotropic cycle into a more elegant and consistent anisotropic symmetrize \rightarrow 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 reintroduce 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_{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_{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. $X_{\text{norm1}} = \text{LayerNorm}(X^{(L-1)})$

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

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

464 10.3 Detailed Computation within the Multi-Head Attention (MHA) Block

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

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

468 10.4 MHA Sub-block 2: Learned Projections for Q, K, and V

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

- 471 • **Weight Matrices:**

- 472 – $W^Q \in \mathbb{R}^{d_{\text{model}} \times d_{\text{model}}}$ (rows \times columns: $d_{\text{model}} \times d_{\text{model}}$)
- 473 – $W^K \in \mathbb{R}^{d_{\text{model}} \times d_{\text{model}}}$ (rows \times columns: $d_{\text{model}} \times d_{\text{model}}$)
- 474 – $W^V \in \mathbb{R}^{d_{\text{model}} \times d_{\text{model}}}$ (rows \times columns: $d_{\text{model}} \times d_{\text{model}}$)

- 475 • **Computation:**

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

- 476 • **Resulting Dimensions:** The resulting matrices Q' , K' , and V' each have the dimension
- 477 $\mathbb{R}^{B \times T \times d_{\text{model}}}$.
- 478 • **Prepare for Multi-Head Processing:** These matrices are then reshaped and transposed to
- 479 split the d_{model} dimension across the H attention heads.

- 480 – $Q = \text{reshape}(Q') \rightarrow \mathbb{R}^{B \times T \times H \times d_k} \xrightarrow{\text{transpose}} \mathbb{R}^{B \times H \times T \times d_k}$
- 481 – $K = \text{reshape}(K') \rightarrow \mathbb{R}^{B \times T \times H \times d_k} \xrightarrow{\text{transpose}} \mathbb{R}^{B \times H \times T \times d_k}$
- 482 – $V = \text{reshape}(V') \rightarrow \mathbb{R}^{B \times T \times H \times d_v} \xrightarrow{\text{transpose}} \mathbb{R}^{B \times H \times T \times d_v}$

483 10.5 MHA Sub-block 1: Scaled Dot-Product Attention

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

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

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

488 The matrix multiplication is performed on the last two dimensions. The dimensions are:
 489 $(\mathbb{R}^{B \times H \times T \times d_k}) \cdot (\mathbb{R}^{B \times H \times d_k \times T}) \rightarrow \mathbb{R}^{B \times H \times T \times T}$. The resulting **Scores** matrix contains,
 490 for each head, a $T \times T$ matrix of similarity scores between all pairs of tokens in the sequence.

- 491 • **Step 2: Scale, Mask, and Softmax:** The scores are scaled to stabilize gradients, masked
- 492 to enforce causality (i.e., a token cannot attend to future tokens), and normalized into a
- 493 probability distribution using softmax.

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

- 494 – The mask $M \in \mathbb{R}^{T \times T}$ is a matrix where $M_{ij} = -\infty$ for $j > i$ (upper triangle) and
- 495 $M_{ij} = 0$ otherwise. This is broadcast across the batch and head dimensions.
- 496 – The softmax is applied along the last dimension (i.e., row-wise for each $T \times T$ matrix).
- 497 – 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: $(\mathbb{R}^{B \times H \times T \times T}) \cdot (\mathbb{R}^{B \times H \times T \times d_v}) \rightarrow \mathbb{R}^{B \times H \times T \times d_v}$.

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.