LORENTZ BUSEMANN LANGUAGE MODELS FOR EFFICIENT HYPERBOLIC LANGUAGE MODELING

Anonymous authors
Paper under double-blind review

ABSTRACT

Prior studies suggest that token level structure is hierarchical and non-Euclidean, motivating hyperbolic large language models (LLMs). Fully hyperbolic models such as HELM add hyperbolic operations throughout the Transformer, increasing training and inference costs. Besides, evidence for hyperbolic structure mainly concerns token embeddings and last hidden states, which together determine the logits at the output layer. We therefore propose the Lorentz Busemann Language Model (LBLM), a hyperbolic LLM that retains standard Transformer backbone and introduces learnable hyperbolic token embeddings at the output layer. LBLM uses the Lorentz Busemann Logit (LBL) to score tokens with the Busemann function, treating hidden states as boundary directions and avoiding their exponential mapping into hyperbolic space. To optimize the token embeddings under learnable curvature, we develop Curvature Learning Riemannian Adam (CRAdam), which keeps the embeddings in the hyperbolic space and preserves the moment norm across curvature changes, without repeated logarithmic and exponential maps through the tangent space. Across scales from 0.3B to 1.4B parameters, LBLM improves mean accuracy by up to 7.8% relative to Euclidean baseline trained with the same recipe, with at most 5.0% training overhead. Compared with the fully hyperbolic HELM, which cannot be trained stably under this recipe, LBLM attains higher accuracy with each model trained under its own recipe. Under identical workloads, LBLM achieves at least 5.3$\times$ the training throughput and 2.8$\times$ the inference throughput of HELM. LBLM offers a simple and efficient hyperbolic LLM, showing that hyperbolic space need not permeate the Transformer backbone to benefit language modeling.

1 INTRODUCTION

Recent work has explored hyperbolic geometry for large language models (LLMs) (Yang et al., 2025; He et al., 2026b), using its capacity to embed hierarchical structure as a geometric inductive bias for token representations (Nickel & Kiela, 2017; Desai et al., 2023). This direction is motivated by evidence that token level structure in language is hierarchical and non-Euclidean. Token frequencies follow power law distributions (He et al., 2025), and token embeddings exhibit hierarchical structure as measured by δ-hyperbolicity (Yang et al., 2025; Gromov, 1987). The token space of LLMs has also been shown to exhibit non-Euclidean structure (Robinson et al., 2024), and last hidden states show predominantly negative Ollivier Ricci curvature (He et al., 2026b; Jost & Liu, 2014). Among these, HELM incorporates hyperbolic geometry throughout the Transformer, demonstrating its potential for language modeling (He et al., 2026b). However, maintaining hyperbolic computation across the backbone requires specialized attention, normalization, and residual operations at every layer (Yang et al., 2024; He et al., 2026b), increasing the cost of training and inference. Notably, the evidence mainly concerns token embeddings and last hidden states, precisely the two quantities from which the output layer computes logits. This motivates us to investigate whether introducing hyperbolic geometry at the output layer can retain its benefits while keeping computation close to that of standard Euclidean Transformers.

Retaining a Euclidean backbone while introducing hyperbolic geometry at the output layer yields a hybrid architecture. Prior hybrid designs map encoder outputs into hyperbolic space via the exponential map and optimize them with task specific objectives (Khrulkov et al., 2020; Madhu et al., 2026). These designs enable reuse of standard architectures and computational components. How-

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

ever, extending this approach to autoregressive language modeling raises challenges. In an LLM, the output layer computes logits from Euclidean hidden states and hyperbolic token embeddings, so the exponential map must be applied to the hidden states at every token position before the entire vocabulary is scored. Doing so adds geometric computation, introduces numerical difficulties under finite precision (Desai et al., 2023; Mishne et al., 2023; Klein et al., 2026), and makes the logits sensitive to how the hidden states are scaled (Guo et al., 2022; He et al., 2026a). We instead introduce the *Lorentz Busemann Language Model* (LBLM), a hyperbolic LLM whose output layer computes logits with the *Lorentz Busemann Logit* (LBL). In LBL, the normalized hidden state defines a boundary direction at infinity, token embedding is a learnable point in hyperbolic space, and the logit is the negative Busemann function. Since the hidden state to Busemann function only through its direction, normalization supplies the required information without exponential mapping. Geometrically, the logits rank tokens by the horosphere on which their embeddings lie.

Training the output layer of LBLM requires updating the token embeddings while keeping them in the hyperbolic space. Treating them as Euclidean parameters would ignore the metric under which LBL is defined, so we optimize them with Riemannian optimization (Bécigneul & Ganea, 2019). We further update the curvature of hyperbolic space during pretraining, allowing the geometry of the embedding space to adapt (Desai et al., 2023; Bdeir et al., 2026). Since both the manifold constraint and the metric depend on the curvature, a curvature update requires adjusting the token embeddings as well as the moment accumulated from previous gradients. Bdeir et al. (2026) address this issue by transferring parameters and momentum through the tangent space at the origin. However, at every curvature update, this procedure maps each token embedding to the tangent space at the origin with the logarithmic map under the old curvature and then back onto the manifold with the exponential map under the new curvature, a round trip repeated for the entire vocabulary. We develop *Curvature Learning Riemannian Adam* (CRAdam), which preserves the spatial coordinates of each token embedding in the Lorentz model and the metric norm of its moment at each curvature update. With the time coordinate recomputed from the spatial coordinates, the token embeddings remain on the Lorentz model under the updated curvature, and neither transition requires logarithmic or exponential maps.

We pretrain LBLM at four scales from 0.3B to 1.4B parameters and compare it with a Euclidean baseline and HELM. We further compare LBL with alternative logits in the output layers, and CRAdam with alternative optimization strategies. Our main contributions are as follows:

- We propose LBLM, a hyperbolic LLM that introduces hyperbolic geometry only at the output layer while retaining a standard Transformer backbone and achieving at least $2.8\times$ the inference throughput of fully hyperbolic HELM.
- We propose the LBL for the output layer. LBL uses normalized hidden states as boundary directions for Busemann function and computes logits over hyperbolic token embeddings without exponential mapping at every token position.
- We develop CRAdam to optimize LBLM's output layer. CRAdam learns the curvature jointly with the hyperbolic token embeddings, and keeps the embeddings valid and preserves the metric norm of the first moment during curvature transitions.
- We evaluate LBLM across different model scales against a Euclidean baseline with relative gains of up to 7.8% at nearly the same training cost. Ablations further support LBL and CRAdam against alternative logits and Riemannian optimizers.

2 Related Work

Hyperbolic Language Models. Hyperbolic geometry has been increasingly incorporated into language models and Transformer architectures to better capture the hierarchical structures inherent in natural language. Early work introduced hyperbolic attention mechanisms to capture latent hierarchical structures (Gulcehre et al., 2019). Subsequent work developed fully hyperbolic neural networks, enabling essential neural operations to be performed directly in hyperbolic space (Chen et al., 2022). Building on these developments, Chen et al. (2024) developed a hyperbolic pretrained language model by adapting the major components of the Transformer to operate in hyperbolic space. Moreover, HypFormer developed a fully hyperbolic Transformer by defining essential Transformer operations directly in hyperbolic space (Yang et al., 2024). HELM further extended this

direction to LLMs by constructing major Transformer components, including attention, positional encoding, normalization, and feedforward networks, in hyperbolic space (He et al., 2026b). These approaches mainly modify part or all of the Transformer backbone to operate in hyperbolic space. In contrast, LBLM preserves the standard Transformer backbone and introduces hyperbolic geometry only in its output layer, providing a simpler and effective alternative for hyperbolic LLM.

Busemann Functions in Hyperbolic Learning. Busemann functions measure position along a boundary direction at infinity. In hyperbolic space, their level sets are horospheres, analogous to Euclidean hyperplanes orthogonal to a fixed direction. This geometric structure has been exploited in several hyperbolic representation learning methods. HoroPCA employs horospherical projections for dimensionality reduction (Chami et al., 2021). Atigh et al. (2021) place class prototypes on the boundary of the Poincaré ball and introduce a Busemann loss to learn embeddings to the prototypes. BMLR generalizes multinomial logistic regression to hyperbolic space using Busemann functions (Chen et al., 2026). In BMLR, Busemann functions replace the Euclidean inner products. In contrast, our LBL uses the input features to determine the boundary directions of the Busemann functions, while representing the output weights as hyperbolic points.

Riemannian Optimization. Riemannian optimization generalizes standard optimization methods from Euclidean space to Riemannian manifolds. Bonnabel (2013) generalized stochastic gradient descent to Riemannian manifolds, and Bécigneul & Ganea (2019) further extended adaptive optimization methods to Riemannian settings. These methods typically assume that the manifold remains fixed during optimization. For hyperbolic manifolds with learnable curvature, changes in curvature require the model parameters and optimizer states to be updated consistently. Recently, Bdeir et al. (2026) proposed an optimization method for jointly learning hyperbolic parameters and curvature, where curvature changes are handled by mapping parameters to the tangent space at the origin via the logarithmic map under the old manifold and then to the new manifold via the exponential map. In contrast, our CRAdam preserves the spatial coordinates of points in the Lorentz model under the updated curvature and transforms the first moment while preserving Riemannian norms.

3 PRELIMINARIES

3.1 LORENTZ MODEL

We use the Lorentz model of n-dimensional hyperbolic space with curvature $-c < 0$, $\mathbb{L}_c^n = \{\boldsymbol{x} \in \mathbb{R}^{n+1} : \langle \boldsymbol{x}, \boldsymbol{x} \rangle_{\mathcal{L}} = -1/c,\ x_0 > 0\}$ (Appendix A). Each point $\boldsymbol{x} \in \mathbb{L}_c^n$ is represented as $\boldsymbol{x} = [x_0, \mathbf{x}_{\mathrm{s}}]^\top$, where $x_0 \in \mathbb{R}$ denotes the time coordinate and $\mathbf{x}_{\mathrm{s}} \in \mathbb{R}^n$ denotes the spatial coordinates. In this paper, $\|\cdot\|$ denotes the Euclidean norm, while the Lorentz inner product is defined as $\langle \boldsymbol{x}, \boldsymbol{y} \rangle_{\mathcal{L}} = -x_0 y_0 + \mathbf{x}_{\mathrm{s}}^\top \mathbf{y}_{\mathrm{s}}$. The associated Lorentz norm is denoted by $\|\cdot\|_{\mathcal{L}} = \sqrt{\langle \cdot, \cdot \rangle_{\mathcal{L}}}$.

For any point $\boldsymbol{x} \in \mathbb{L}_c^n$, the tangent space $\mathcal{T}_{\boldsymbol{x}} \mathbb{L}_c^n$ is isomorphic to $\mathbb{R}^n$ and is given by $\mathcal{T}_{\boldsymbol{x}} \mathbb{L}_c^n = \left\{ \boldsymbol{v} \in \mathbb{R}^{n+1} : \langle \boldsymbol{v}, \boldsymbol{x} \rangle_{\mathcal{L}} = 0 \right\}$. For two points $\boldsymbol{x}, \boldsymbol{y} \in \mathbb{L}_c^n$, their geodesic distance is given by $d_{\mathbb{L}}^c(\boldsymbol{x}, \boldsymbol{y}) = 1/\sqrt{c} \cosh^{-1}\left(-c \langle \boldsymbol{x}, \boldsymbol{y} \rangle_{\mathcal{L}}\right)$. Given $\boldsymbol{x} \in \mathbb{L}_c^n$ and a tangent vector $\boldsymbol{v} \in \mathcal{T}_{\boldsymbol{x}} \mathbb{L}_c^n$, $\exp_{\boldsymbol{x}}^c(\boldsymbol{v})$ maps $\boldsymbol{v}$ to a point on $\mathbb{L}_c^n$ along the geodesic starting at $\boldsymbol{x}$ in the direction of $\boldsymbol{v}$. Conversely, for $\boldsymbol{y} \in \mathbb{L}_c^n$, the logarithmic map $\log_{\boldsymbol{x}}^c(\boldsymbol{y})$ sends $\boldsymbol{y}$ to the tangent space at $\boldsymbol{x}$.

3.2 BUSEMANN FUNCTIONS AND HOROSPHERES

In a geodesic metric space, Busemann functions associated with geodesic rays characterize the asymptotic distance of points along a given direction at infinity (Busemann, 2005). Let $\mathcal{M}$ be a geodesic space, and let $\gamma : [0, \infty) \to \mathcal{M}$ be a unit geodesic ray. The ray γ determines an endpoint on the boundary at infinity $\partial_\infty \mathcal{M}$, which specifies its boundary direction, denoted by $\boldsymbol{\xi}$. For any $\boldsymbol{x} \in \mathcal{M}$, the Busemann function associated with γ is defined as $b_\gamma(\boldsymbol{x}) = \lim_{t \to \infty} \left(d(\boldsymbol{x}, \gamma(t)) - t \right)$, where $d(\cdot, \cdot)$ is the geodesic distance in $\mathcal{M}$. Under this sign convention, $b_\gamma(\gamma(s)) = -s$, and $-b_\gamma(\boldsymbol{x})$ can be interpreted as a coordinate of $\boldsymbol{x}$ along $\boldsymbol{\xi}$ (Chami et al., 2021). The boundary direction $\boldsymbol{\xi}$ is represented by a vector with $\langle \boldsymbol{\xi}, \boldsymbol{\xi} \rangle_{\mathcal{L}} = 0$ and $\xi_0 > 0$, which we normalize as $\boldsymbol{\xi} = [1, \mathbf{u}]^\top$ with $\|\mathbf{u}\| = 1$. With this normalization and γ starting at the pole (also referred to as the origin) $\boldsymbol{o} = [1/\sqrt{c}, \mathbf{0}]^\top$, the Busemann function has the closed form (see Appendix B.3 for the detailed derivation): $b_{\boldsymbol{\xi}}(\boldsymbol{x}) = 1/\sqrt{c} \log\left(-\sqrt{c} \langle \boldsymbol{x}, \boldsymbol{\xi} \rangle_{\mathcal{L}}\right)$.

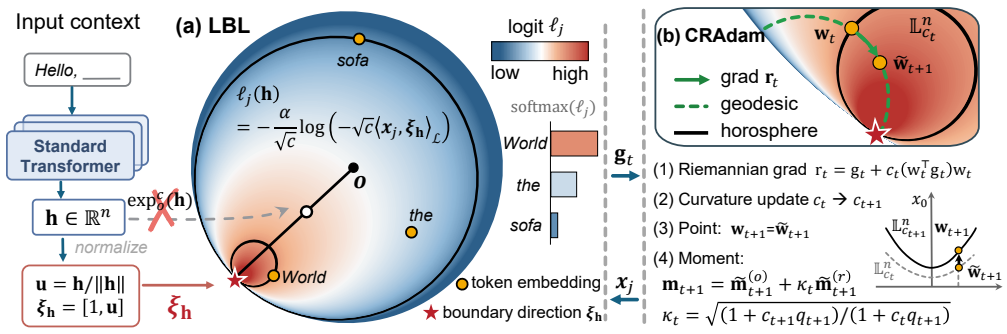


Figure 1: Overview of LBLM. (a) LBL turns the hidden state into boundary direction $\boldsymbol{\xi}_\mathbf{h} = [1, \mathbf{h}/\|\mathbf{h}\|]$. Token embedding $\boldsymbol{x}_j \in \mathbb{L}_c^n$ is scored by $\ell_j(\mathbf{h}) = -\alpha\, b_{\boldsymbol{\xi}_\mathbf{h}}(\boldsymbol{x}_j)$. (b) CRAdam keeps the spatial coordinates of embedding and rescales the first moment at each curvature update.

For a fixed $\boldsymbol{\xi} \in \partial_\infty \mathbb{L}_c^n$ and $\tau \in \mathbb{R}$, the level set $\mathcal{H}_{\boldsymbol{\xi},\tau} = \{\boldsymbol{x} \in \mathbb{L}_c^n : b_{\boldsymbol{\xi}}(\boldsymbol{x}) = \tau\}$ is called a horosphere (Heintze & Im Hof, 1977). Points on the same horosphere share the same Busemann value with respect to $\boldsymbol{\xi}$. The family $\{\mathcal{H}_{\boldsymbol{\xi},\tau}\}_{\tau\in\mathbb{R}}$ organizes the space according to position along the direction represented by $\boldsymbol{\xi}$, with smaller τ corresponding to horospheres closer to $\boldsymbol{\xi}$. Changing the starting point of γ preserves this family and shifts τ by a constant.

4 Method

4.1 Lorentz Busemann Logit (LBL)

Let $\mathbf{h} \in \mathbb{R}^n$ denote the final hidden state at a given sequence position, and let each vocabulary token $j \in \mathcal{V}$ be a learnable point $\boldsymbol{x}_j = [x_{j,0}, \mathbf{w}_j]^\top \in \mathbb{L}_c^n$. This construction creates an apparent geometric mismatch: the contextual representation $\mathbf{h} \in \mathbb{R}^n$ is produced by the Transformer, whereas the vocabulary representations lie in the hyperbolic space (or Lorentz model). A natural way to bridge this mismatch is to lift $\mathbf{h} \in \mathbb{R}^n$ to the tangent space at the pole $\boldsymbol{o}$ and map it onto the Lorentz space via the exponential map, $\boldsymbol{z}_\mathbf{h} = \exp_\boldsymbol{o}^c\left([0, \mathbf{h}]^\top\right) \in \mathbb{L}_c^n$, before computing the vocabulary logits. It is a common approach to introduce hyperbolic representations into Euclidean neural networks (He et al., 2026a; Chen et al., 2026; Yang et al., 2025). LBL requires only the direction of $\mathbf{h}$.

Hidden States as Boundary Directions. Writing the normalized direction as $\mathbf{u} = \mathbf{h}/\|\mathbf{h}\|$, the exponential map at the pole $\boldsymbol{o}$ takes the form

$$\exp_\boldsymbol{o}^c([0, \mathbf{h}]) = 1/\sqrt{c}\left[\cosh(\sqrt{c}\|\mathbf{h}\|),\ \sinh(\sqrt{c}\|\mathbf{h}\|)\mathbf{u}\right] = \gamma_\mathbf{u}(\|\mathbf{h}\|), \tag{1}$$

where $\gamma_\mathbf{u}(t) \in \mathbb{L}_c^n$ denotes the geodesic ray starting from $\boldsymbol{o}$ with spatial direction $\mathbf{u}$. Eq. 1 shows that $\|\mathbf{h}\|$ determines only a finite position along the geodesic ray $\gamma_\mathbf{u}$, whereas the spatial direction $\mathbf{u}$ determines the geodesic ray $\gamma_\mathbf{u}$ and its boundary direction $\boldsymbol{\xi}_\mathbf{h} = [1, \mathbf{u}]^\top$ (Appendix B.1). Since the Busemann function $b_{\boldsymbol{\xi}}(\boldsymbol{x})$ depends on the boundary direction $\boldsymbol{\xi}_\mathbf{h}$ rather than on a particular finite point $\boldsymbol{z}_\mathbf{h}$ along the geodesic ray $\gamma_\mathbf{u}$, explicitly constructing $\boldsymbol{z}_\mathbf{h}$ is unnecessary for LBL.

Logits Formation. Given the boundary direction $\boldsymbol{\xi}_\mathbf{h}$ determined by the hidden state and the token embedding $\boldsymbol{x}_j \in \mathbb{L}_c^n$ associated with vocabulary token j, we define LBL as

$$\ell_j(\mathbf{h}) = -\alpha b_{\boldsymbol{\xi}_\mathbf{h}}(\boldsymbol{x}_j) = -\alpha/\sqrt{c}\log\left(-\sqrt{c}\,\langle \boldsymbol{x}_j, \boldsymbol{\xi}_\mathbf{h}\rangle_\mathcal{L}\right), \tag{2}$$

where $\alpha > 0$ is a shared learnable scale. The argument of the logarithm in Eq. 2 is strictly positive for any $\boldsymbol{x}_j \in \mathbb{L}_c^n$ and unit spatial direction $\mathbf{u}$, as established in Corollary 2. The resulting logits are converted into next-token probabilities using the softmax.

LBLM retains the standard Transformer backbone, with hyperbolic token embeddings $\boldsymbol{x}_j$ and only a normalization of hidden states, $\mathbf{u} = \mathbf{h}/\|\mathbf{h}\|$. This gives LBL two properties in both training and inference. First, no hidden state is mapped by the exponential map, which avoids the $\cosh(\cdot)$ and $\sinh(\cdot)$ evaluations in Eq. 1 and their numerical issues under finite precision (Mishne et al., 2023); apart from normalization, the dominant cost is a single matrix product between the normalized hidden states and the spatial coordinates of the token embeddings, with the same shape as in a

Euclidean output layer. Second, since the logits are invariant to $\|\mathbf{h}\|$, there is no need to determine how Euclidean features should be scaled before the exponential map when adapting to different models.

Relation to BMLR. BMLR (Chen et al., 2026) also constructs logits by evaluating a Busemann function at a hyperbolic feature point. Each token defines a boundary direction $\boldsymbol{\xi}_j = [1, \mathbf{w}_j/\|\mathbf{w}_j\|]$, and the logit $-\alpha_j b_{\boldsymbol{\xi}_j}(\boldsymbol{z}_\mathbf{h}) + e_j$ is evaluated at the mapped hidden state $\boldsymbol{z}_\mathbf{h} = \exp^c_{\boldsymbol{o}}([0, \mathbf{h}])$. In an LLM output layer, this requires mapping every hidden state into hyperbolic space. Moreover, since the exponential map places $\boldsymbol{z}_\mathbf{h}$ along $\gamma_\mathbf{u}$ based on $\|\mathbf{h}\|$ (Eq. 1), the logits of BMLR depend on how $\mathbf{h}$ is scaled before the mapping. By letting the hidden state define the boundary direction, LBL avoids exponential map and the choice of scaling for $\mathbf{h}$. Section 5.2.1 confirms that LBL attains higher accuracy than BMLR, while requiring less memory and achieving higher throughput when both are implemented directly from their equations.

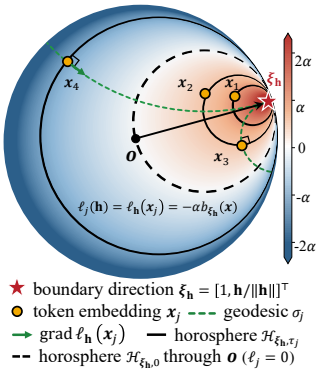


Figure 2: Illustration of LBL.

As illustrated in Figure 2, we view LBL as a function of the token embedding, $\ell_\mathbf{h}(\boldsymbol{x}) = -\alpha b_{\boldsymbol{\xi}_\mathbf{h}}(\boldsymbol{x})$, with $\ell_j(\mathbf{h}) = \ell_\mathbf{h}(\boldsymbol{x}_j)$. Its level sets are horospheres $\mathcal{H}_{\boldsymbol{\xi}_\mathbf{h},\tau_j}$ centered at $\boldsymbol{\xi}_\mathbf{h}$, where $\tau_j = b_{\boldsymbol{\xi}_\mathbf{h}}(\boldsymbol{x}_j)$ is the Busemann value. Embeddings on the same horosphere share the logit $-\alpha\tau_j$, which increases as the horosphere gets closer to $\boldsymbol{\xi}_\mathbf{h}$. The $\operatorname{grad}\ell_\mathbf{h}(\boldsymbol{x}_j) \in \mathcal{T}_{\boldsymbol{x}_j}\mathbb{L}^n_c$ of $\ell_\mathbf{h}$ at $\boldsymbol{x}_j$ points along the geodesic σ_j from $\boldsymbol{x}_j$ to $\boldsymbol{\xi}_\mathbf{h}$, orthogonally to $\mathcal{H}_{\boldsymbol{\xi}_\mathbf{h},\tau_j}$ (Theorem 5).

4.2 Riemannian Optimization

Since the Riemannian gradient of LBL points along geodesics (Section 4.1), we update the token embeddings with CRAdam, a Riemannian optimizer that keeps them on the Lorentz manifold as the curvature is learned. CRAdam stores the spatial coordinates $\mathbf{w}_j$ of $\boldsymbol{x}_j = [x_{j,0}, \mathbf{w}_j]$ and reconstructs $x_{j,0}$, giving each token the same number of learnable parameters as a Euclidean output layer. Omitting the token index, let $\boldsymbol{x} = [x_0, \mathbf{w}]$. As in Algorithm 1, CRAdam first performs a Riemannian Adam update under the current curvature c_t (Bécigneul & Ganea, 2019), producing the intermediate point $\widetilde{\mathbf{w}}_{t+1}$, transported first moment $\widetilde{\mathbf{m}}_{t+1}$, and updated second moment v_t^+.

Algorithm 1: One CRAdam step for an embedding

Require: Spatial coordinates $\mathbf{w}_t$, first moment $\mathbf{m}_t$, second moment v_t, log-curvature ω_t, and gradients $\mathbf{g}_t$ and $g_{\omega,t}$

$c_t \leftarrow \exp(\omega_t)$
Riemannian update under c_t
$\mathbf{r}_t \leftarrow \mathbf{g}_t + c_t(\mathbf{w}_t^\top \mathbf{g}_t)\mathbf{w}_t$ ▷ Riemannian gradient
$(\mathbf{m}_t^+, v_t^+) \leftarrow \mathrm{AdamMoments}(\mathbf{m}_t, v_t, \mathbf{r}_t)$
$\widetilde{\mathbf{w}}_{t+1} \leftarrow \mathrm{RiemannianStep}_{c_t}(\mathbf{w}_t, \mathbf{m}_t^+, v_t^+)$
$\widetilde{\mathbf{m}}_{t+1} \leftarrow \mathrm{PT}^{c_t}_{\mathbf{w}_t \to \widetilde{\mathbf{w}}_{t+1}}(\mathbf{m}_t^+)$
Curvature update and point transition
$\omega_{t+1} \leftarrow \mathrm{CurvUpdate}(\omega_t, g_{\omega,t})$
$c_{t+1} \leftarrow \exp(\omega_{t+1})$
$\mathbf{w}_{t+1} \leftarrow \widetilde{\mathbf{w}}_{t+1}$
$x_{0,t+1} \leftarrow \sqrt{1/c_{t+1} + \|\mathbf{w}_{t+1}\|^2}$
First moment transition
$q_{t+1} \leftarrow \|\mathbf{w}_{t+1}\|^2$
$\kappa_t \leftarrow \sqrt{(1 + c_{t+1}q_{t+1})/(1 + c_t q_{t+1})}$
$\mathbf{m}_{t+1} \leftarrow \widetilde{\mathbf{m}}^{(o)}_{t+1} + \kappa_t \widetilde{\mathbf{m}}^{(r)}_{t+1}$
$v_{t+1} \leftarrow v_t^+$
return $\mathbf{w}_{t+1}, \mathbf{m}_{t+1}, v_{t+1}, \omega_{t+1}$

Point Transition under Updated Curvature. We parameterize the curvature as $\omega_t = \log c_t$, so that $c_t = \exp(\omega_t) > 0$ (Desai et al., 2023). After updating the curvature to c_{t+1}, we preserve the spatial coordinates $\widetilde{\mathbf{w}}_{t+1}$ obtained from the Riemannian update under c_t, setting $\mathbf{w}_{t+1} = \widetilde{\mathbf{w}}_{t+1}$, and reconstruct the time coordinate as $x_{0,t+1} = \sqrt{1/c_{t+1} + \|\mathbf{w}_{t+1}\|^2}$, which ensures that the updated embedding lies on $\mathbb{L}^n_{c_{t+1}}$. This choice follows a simple principle: the quantity preserved during a curvature transition should be the one held fixed when the curvature gradient is computed. Since x_0 is recomputed from $\mathbf{w}$ and c in the forward pass, the curvature gradient $g_{\omega,t}$ is taken with $\mathbf{w}$ fixed, and preserving $\mathbf{w}$ lets the curvature update act on the model as this gradient assumes. In contrast, transferring each embedding through the tangent space at the origin (Bdeir et al., 2026) preserves its distance from the origin rather than its spatial coordinates, which departs from this principle and requires logarithmic and exponential maps (Appendix D.3.1).

First Moment Transition. The first moment accumulates information from previous gradients and directly affects the adaptive Riemannian update. When the curvature changes, however, the metric also changes, which can alter the metric norm of this accumulated state even without a new gradient

Table 1: Pretraining results over 3 runs. Relative **gains** over baseline are reported. Best are in **bold**.

Method	Task Performance							Metrics	
	ARC-E	ARC-C	COPA	CSQA	OBQA	BoolQ	BB-Lang	Avg. (↑)	Gain (↑)
Depth 12, 0.3B Parameters									
Euclidean	51.5	25.2	54.7	34.1	28.0	52.1	25.6	38.75 ± 0.84	–
LBLM (ours)	**56.9**	**28.6**	**59.0**	**34.5**	**31.9**	**55.9**	**25.7**	**41.78** ± 0.35	+7.8%
Depth 16, 0.5B Parameters									
Euclidean	57.5	28.9	60.3	**30.9**	32.4	58.7	25.5	42.03 ± 0.63	–
LBLM (ours)	**62.3**	**33.0**	**62.0**	28.0	**35.5**	**59.1**	**25.9**	**43.69** ± 0.17	+4.0%
Depth 20, 0.9B Parameters									
Euclidean	63.4	33.1	64.7	21.8	37.3	54.2	**25.6**	42.86 ± 0.41	–
LBLM (ours)	**66.5**	**36.5**	**66.7**	**26.4**	**38.6**	**55.8**	25.3	**45.11** ± 0.54	+5.3%
Depth 24, 1.4B Parameters									
Euclidean	68.1	35.4	**66.3**	28.1	37.9	54.6	25.4	45.12 ± 1.04	–
LBLM (ours)	**71.3**	**40.2**	66.0	**32.5**	**40.2**	**56.9**	**25.5**	**47.51** ± 0.58	+5.3%

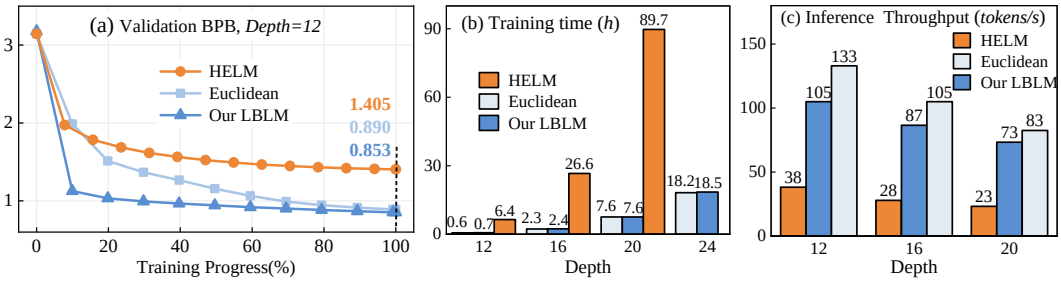


Figure 3: Performance and efficiency comparison. (a) Validation BPB. (b) Training time across model depths; see Table 10 for details. (c) Inference throughput across model depths.

contribution. We therefore choose to preserve the metric norm of the accumulated first moment across curvature changes. This maintains the accumulated optimization state from the first moment under the new curvature c_{t+1}, so that the curvature change itself does not rescale its metric norm. The empirical effect of this choice is evaluated in Table 6. Let $q_{t+1} = \|\mathbf{w}_{t+1}\|^2$, and let $\widetilde{\mathbf{m}}_{t+1}^{(r)}$ and $\widetilde{\mathbf{m}}_{t+1}^{(o)}$ denote the components parallel and orthogonal to $\mathbf{w}_{t+1}$, respectively. The exact transition is

$$\kappa_t = \sqrt{(1 + c_{t+1}q_{t+1})/(1 + c_t q_{t+1})}, \quad \mathbf{m}_{t+1} = \widetilde{\mathbf{m}}_{t+1}^{(o)} + \kappa_t \widetilde{\mathbf{m}}_{t+1}^{(r)}. \tag{3}$$

When $c_{t+1} = c_t$, we have $\kappa_t = 1$, and both the point and moment transitions reduce to the identity, recovering the fixed curvature Riemannian Adam update. The details are in Appendix D.

5 Experiments

5.1 Main Results

Comparison with Euclidean baseline. Table 1 compares LBLM with the Euclidean baseline across four model scales. LBLM improves mean accuracy at every scale, with relative gains of 4.0–7.8%, and achieves lower validation BPB throughout training (Figures 3a, 7a, and 8). These gains come with nearly the same training time as the Euclidean baseline (Figure 3b). Experimental settings are provided in Appendix E.1.

Comparison with HELM. We implement HELM with the same depth, model dimension, number of attention heads, and token allocation rule as the Euclidean baseline and LBLM. Because its Lorentz layers introduce additional parameters, HELM has more scaling parameters and therefore receives more training tokens under the shared allocation rule. HELM differs from the standard Transformer by using squared Lorentz distances $d_c^2(\cdot)$ for attention and Lorentz geometric opera-

tions throughout the remaining components (He et al., 2026b). Stable training of HELM architecture depends on settings that its authors established for it, and a training recipe developed for a Euclidean backbone is not a neutral choice. In our experiments, HELM does not train stably under the recipe used for the Euclidean baseline and LBLM: its validation BPB fails to decrease and training becomes unstable (Figure 7b). We therefore use the recipe released by He et al. (2026b). This follows the usual practice of comparing complete methods under their respective recommended configurations. As a consequence, comparisons between LBLM and HELM reflect differences in both method and recipe. To verify our HELM reimplementation, we also train a Euclidean backbone with the HELM recipe at depth 12. HELM reaches 32.53 against 30.17 for this backbone (Table 2), reproducing the direction of the improvement reported by He et al. (2026b). This check is performed only at depth 12 because it serves to validate the implementation rather than characterize HELM scaling. The comparison also shows that the training recipe substantially affects absolute accuracy: the Euclidean backbone achieves 30.17 under the HELM recipe versus 38.75 under ours.

Table 2: Gains over matched Euclidean baselines at depth 12. Δ and ΔT denote the absolute accuracy gain and relative training time to the baselines. Results are averaged over 3 runs.

Recipe	Model	Tokens(B)	Avg. (↑)	Δ	ΔT
HELM	Euc.	1.32	30.17±0.38	–	–
	HELM	1.67	32.53±1.39	+2.36	∼10×
Shared	Euc.	1.32	38.75±0.84	–	–
	LBLM	1.32	**41.78**±0.35	+3.03	+5.0%

Accuracy. We compare LBLM and HELM as complete methods, each trained with its recipe. As shown in Table 3, LBLM achieves higher average accuracy at all three depths despite HELM using more training tokens; since the recipes differ, we attribute this to method and recipe jointly rather than to architecture alone. Nevertheless, it shows that, when each method is trained with its own recipe, LBLM delivers higher accuracy in practice.

Cost of the improvement. As shown in Table 2, both HELM and LBLM improve over their baselines, but at very different cost. LBLM gains 3.03 points over its Euclidean baseline with the same number of training tokens (1.32B) and only 5% more training time ΔT. HELM gains 2.36 points but uses more scaling parameters (0.14B vs. 0.11B), 1.27× the training tokens (1.67B vs. 1.32B), and about 10× the training time of its baseline. Moreover, even with this improvement, HELM remains below both the Euclidean baseline trained with our recipe and LBLM.

Table 3: Results over 3 runs, except HELM at depth 20. Detailed results in Table 12.

Method	Depth 12	Depth 16	Depth 20
HELM	32.53 ± 1.39	33.70 ± 1.55	34.84
LBLM	**41.78** ± 0.35	**43.69** ± 0.17	**45.11** ± 0.54

Training and inference cost. LBLM confines hyperbolic operations to the output layer, whereas HELM applies them at every Transformer layer. As a result, LBLM is several times faster than HELM in both training and inference. For training, we report HELM's training time in Figure 3b but do not compare it directly with that of LBLM, since HELM is trained with more tokens and a smaller device batch size (Table 9). Instead, we compare training throughput under an identical workload, running LBLM at HELM's device batch size: LBLM achieves 5.3–6.3× the throughput of HELM at all three depths (Figure 4). This comparison is conservative for LBLM, which in actual training uses a larger device batch size and attains even higher throughput (Figure 5). For inference, under identical decoding settings, LBLM achieves 2.8–3.2× the throughput of HELM (Figure 3c).

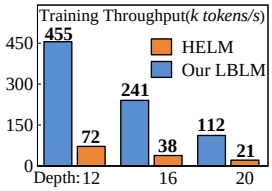


Figure 4: Comparison of training throughput.

Relative to the Euclidean baseline, which shares its recipe, token budget, and device batch size, LBLM adds only modest overhead. In training, LBLM retains 94–99% of the Euclidean baseline's throughput under the same batch size; accordingly, its training time is about 5% longer at depth 12 and within 2% at depths 20 and 24 (Table 10). At inference, it retains 79–88% of the baseline's throughput, whereas HELM retains less than 30% (Figure 3c). In both training and inference, the overhead of LBLM decreases with depth, because the cost of its output layer grows more slowly than that of the backbone, suggesting that it shrinks further for larger models. LBLM attains higher accuracy than HELM when each is trained with its own recipe, improves over the Euclidean base-

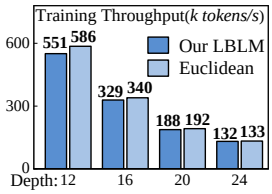


Figure 5: Comparison of training throughput.

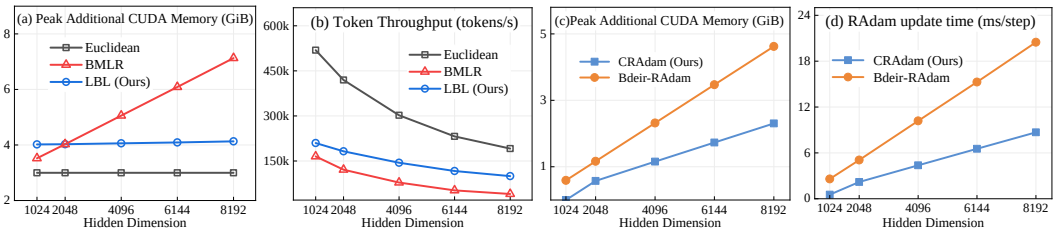


Figure 6: Computational efficiency across hidden dimensions. (a)–(b) compare the Euclidean inner product, BMLR, and LBL in peak additional CUDA memory and forward throughput. (c)–(d) compare memory and update time for our CRAdam and Bdeir-RAdam.

Table 4: Comparison with BMLR over 3 runs. BMLR-PM denotes the BMLR variant with parameters matched to LBL by removing the class specific scale α_j and bias e_j terms. $\phi(\mathbf{h})$ is the transformation applied to the hidden state before the exponential map.

Method / $\phi(\mathbf{h})$	**Task Performance**							**Avg. (↑)**
	ARC-E	**ARC-C**	**COPA**	**CSQA**	**OBQA**	**BoolQ**	**BB-Lang**	
BMLR-PM ($\mathbf{h}$)	33.4	22.5	47.3	20.0	24.9	52.4	25.2	32.24 $\pm$ 0.57
BMLR-PM ($\mathbf{h}/\sqrt{n}$)	54.9	27.5	56.3	30.1	**32.5**	53.4	25.1	39.98 $\pm$ 0.67
BMLR ($\mathbf{h}$)	48.3	25.5	47.7	30.6	28.4	52.7	25.6	36.96 $\pm$ 0.94
BMLR ($\mathbf{h}/\|\mathbf{h}\|$)	55.9	26.8	56.0	26.5	32.1	54.6	25.1	39.57 $\pm$ 0.55
LBL (ours)	**56.9**	**28.6**	**59.0**	**34.5**	31.9	**55.9**	**25.7**	**41.78** $\pm$ 0.35

line at nearly the same training cost, and runs several times faster than HELM in both training and inference. See Appendix G for details.

5.2 Ablation Studies

5.2.1 Comparison of Logit Formulations

Comparison with BMLR. Table 4 compares LBL with BMLR (Chen et al., 2026) and BMLR-PM. Since BMLR applies the exponential map to the hidden state, its accuracy depends on how $\mathbf{h}$ is scaled before the mapping: applying the map to the unscaled $\mathbf{h}$ gives 32.24 for BMLR-PM and 36.96 for BMLR, whereas the best scaling raises them to 39.98 and 39.57, respectively. LBL achieves higher average accuracy than both BMLRs without requiring such a scaling choice. LBL is also more efficient: it achieves higher token throughput, and its peak additional memory remains nearly constant as the hidden dimension grows (Figures 6a and 6b). This is because LBL neither applies the exponential map to hidden states nor normalizes the full vocabulary matrix $\mathbf{W} \in \mathbb{R}^{|\mathcal{V}|\times n}$ to construct boundary directions, so its direct form is already efficient, whereas BMLR would require a reformulated implementation to reduce this overhead. These suggest that using LBL as the logit is both more accurate and, in its direct form, more efficient than BMLR. See Appendix H.1 for details.

Further Comparison with Alternative Output Layers. We compare LBL with two types of alternative logits: Euclidean alternatives to the inner product logits, and hyperbolic logits that replace LBL within LBLM. Table 5 evaluates whether Euclidean inner product logits with a learnable scale α (variant II) or negative Euclidean geodesic distance logits (variant III) can match LBLM's performance. For the hyperbolic geodesic variant IV, we replace LBL in LBLM with negative geodesic distance logits. Adding scale α does not close the average accuracy gap to LBLM. The Euclidean geodesic distance variant also achieves lower average accuracy than LBLM. Furthermore, replacing LBL within LBLM with hyperbolic geodesic distance logits reduces average accuracy, supporting the choice of LBL within LBLM. See Appendix H.2 for details.

Table 5: Comparison of different logits. See Table 14 for details.

Variant	**Avg. (↑)**
I	38.94
II	38.83
III	35.81
IV	39.56
Ours	**42.24**

Table 6: Comparison of optimization strategies. The second block is evaluated over 3 runs. *CRAdam w/o MT* removes the moment transition during curvature updates, keeping $\mathbf{m}_{t+1} = \widetilde{\mathbf{m}}_{t+1}$.

Method	c Update	Task Performance							Avg. (↑)
		ARC-E	ARC-C	COPA	CSQA	OBQA	BoolQ	BB-Lang	
Adam	No	47.6	25.9	47.0	22.6	**29.2**	56.1	24.9	36.19
Adam	Yes	**48.8**	**27.6**	**51.0**	**24.2**	27.2	**60.0**	**25.0**	**37.69**
Bdeir-RAdam	Yes	54.7	28.2	58.3	30.9	31.7	**56.1**	25.5	40.78 ± 0.43
CRAdam	No	55.5	27.7	57.7	34.3	31.7	52.9	25.5	40.75 ± 0.96
CRAdam w/o MT	Yes	55.4	27.4	57.7	29.4	**33.2**	54.0	25.7	40.40 ± 0.40
CRAdam	Yes	**56.9**	**28.6**	**59.0**	**34.5**	31.9	55.9	**25.7**	**41.78** ± 0.35

5.2.2 Effect of Riemannian Optimization

Ablation of CRAdam. Table 6 ablates the components of CRAdam. For learnable curvature under Adam, we preserve the spatial coordinates of the Lorentz token embeddings as in CRAdam. Learning the curvature improves the average score under Adam, and under CRAdam when combined with the first moment transition; under fixed curvature, CRAdam improves over Adam by 4.56 points. Since LBL is defined in the Lorentz model, whereas Adam updates the spatial coordinates as if they were Euclidean, we attribute the lower scores under Adam to the mechanism in the spatial coordinates analyzed in Appendix D.1. In these experiments, learning the curvature without adapting the first moment shows no clear improvement over fixed curvature CRAdam. CRAdam w/o MT keeps the first moment unchanged, allowing its metric norm to change during curvature updates (Appendix D.3.3); its average score is lower than fixed curvature CRAdam. With the proposed first moment transition, which preserves this metric norm, CRAdam achieves the highest average score.

Comparison with Bdeir-RAdam. Bdeir-RAdam (Bdeir et al., 2026) is an existing RAdam for learnable curvature, and we compare it with CRAdam under the same output layer. With LBL, CRAdam outperforms Bdeir-RAdam in average accuracy over 3 runs (Table 6). Table 7 further shows that CRAdam outperforms Bdeir-RAdam (B-RAdam) under BMLR, hyperbolic geodesic logits (variant IV), and LBL, indicating that its advantage is not specific to LBL. Moreover, our LBL achieves the highest score under both RAdam optimizers. Together with the lower update time and peak additional memory in Figures 6c and 6d, these results support the effectiveness and efficiency of the complete CRAdam design in the evaluated settings. Details are given in Appendix H.3.

Table 7: Comparison of Bdeir-RAdam and CRAdam across different logits. Details are in Table 16.

Logits	B-RAdam	CRAdam
BMLR	33.43	**38.29** ↑ 4.86
Variant IV	38.64	**39.56** ↑ 0.92
LBL (ours)	40.17	**42.24** ↑ 2.07

6 Conclusion

We introduced LBLM, a hyperbolic LLM that introduces hyperbolic geometry at the output layer. Its output layer computes logits with LBL, which uses normalized hidden states as boundary directions for Busemann function, and is optimized with CRAdam, which learns the curvature jointly with the token embeddings. Across scales from 0.3B to 1.4B parameters, LBLM outperforms a Euclidean baseline trained with the same recipe and number of tokens, with at most 5.0% training overhead. HELM cannot be trained stably under this recipe; with each trained under its own recipe, LBLM attains higher accuracy than HELM, and under identical workloads it achieves several times HELM's training and inference throughput. Motivated by the hierarchical structure of language, LBLM shows through its downstream accuracy, validation BPB, and efficiency that introducing hyperbolic geometry at the output layer provides an effective and efficient alternative to making the entire Transformer hyperbolic. Finally, since LBL depends on hidden states only through their directions and requires no exponential map, it may offer a convenient way to introduce hyperbolic geometry into pretrained Euclidean LLMs, which we leave to future work.

AI USE STATEMENT

In this work, we used generative AI tools for assisting in writing parts of the code for our proposed method and improving the readability and language of the manuscript. We have not used generative AI tools for proposing or refining hypotheses, designing or providing feedback on research methodology or experiments, developing theoretical models or conceptual frameworks, cleaning or reformatting datasets, generating synthetic datasets, supporting qualitative or thematic data analysis, interpreting results, formulating mathematical claims, providing critical ingredients for or assisting in the writing of proofs, or translation. We have reviewed all AI-assisted work. All AI-assisted code was reviewed and tested for correctness by authors, and all experiments were run by the authors. The paper was drafted by the authors, and AI-polished text was checked by the authors to ensure that its technical content and claims were not altered. We take responsibility for the final content of this work, including text, claims or artifacts produced with the aid of generative AI.

REPRODUCIBILITY STATEMENT

We have made efforts to ensure the reproducibility of our proposed LBLM. Our proposed LBL is defined in Section 4.1, and our proposed CRAdam is described in Section 4.2 and Algorithm 1. For the theoretical results, Appendix A reviews the Lorentz model, Appendices B.1 and B.3 derive the closed-form Busemann function and the boundary direction determined by a hidden state, Corollary 2 shows that LBL is well defined, and Theorem 5 characterizes its Riemannian gradient; complete derivations of the point and moment transitions of CRAdam are given in Appendix D. Appendix E specifies the codebase, training data, evaluation benchmarks, the random seeds used, and training hyperparameters, while Appendix E.2 lists the model configurations, training token budgets, and device batch sizes of all models in Table 9. Appendix F describes the implementation of LBL, including the fused Triton loss kernel, the custom backward pass, and the fixed-order reductions used for the gradients of the shared scale and the curvature. For the compared methods, Appendix G details our reimplementation of HELM; Appendix H.1 describes the implementations of BMLR and BMLR-PM and the hidden state scaling choices; Appendix H.2 defines the alternative logit variants in Table 15; and Appendix H.3.2 describes our implementation of Bdeir-RAdam. The protocols of all efficiency measurements, including hardware and the definitions of throughput and peak additional memory, are given in Appendix G for end-to-end training and inference, Appendix H.1.1 for the output-layer benchmark, and Appendix H.3.3 for the optimizer benchmark.

REFERENCES

Mina Ghadimi Atigh, Martin Keller-Ressel, and Pascal Mettes. Hyperbolic Busemann learning with ideal prototypes. In *Advances in Neural Information Processing Systems*, 2021.

Ahmad Bdeir, Johannes Burchert, Lars Schmidt-Thieme, and Niels Landwehr. Robust hyperbolic learning with curvature-aware optimization. *Advances in Neural Information Processing Systems*, 38:81859–81884, 2026.

Gary Bécigneul and Octavian-Eugen Ganea. Riemannian adaptive optimization methods. In *International Conference on Learning Representations*, 2019.

BIG-bench authors. Beyond the imitation game: Quantifying and extrapolating the capabilities of language models. *Transactions on Machine Learning Research*, 2023. ISSN 2835-8856.

Silvere Bonnabel. Stochastic gradient descent on Riemannian manifolds. *IEEE Transactions on Automatic Control*, 58(9):2217–2229, 2013.

Martin R Bridson and André Haefliger. *Metric spaces of non-positive curvature*. Springer Science & Business Media, 2013.

Herbert Busemann. *The geometry of geodesics*. Courier Corporation, 2005.

James W Cannon, William J Floyd, Richard Kenyon, Walter R Parry, et al. Hyperbolic geometry. *Flavors of geometry*, 31(59-115):2, 1997.

Ines Chami, Zhitao Ying, Christopher Ré, and Jure Leskovec. Hyperbolic graph convolutional neural networks. *Advances in Neural Information Processing Systems*, 32, 2019.

Ines Chami, Albert Gu, Dat P Nguyen, and Christopher Ré. Horopca: Hyperbolic dimensionality reduction via horospherical projections. In *International Conference on Machine Learning*, pp. 1419–1429. PMLR, 2021.

Weize Chen, Xu Han, Yankai Lin, Hexu Zhao, Zhiyuan Liu, Peng Li, Maosong Sun, and Jie Zhou. Fully hyperbolic neural networks. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics*, pp. 5672–5686, 2022.

Weize Chen, Xu Han, Yankai Lin, Kaichen He, Ruobing Xie, Jie Zhou, Zhiyuan Liu, and Maosong Sun. Hyperbolic pre-trained language model. *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, 32:3101–3112, 2024.

Ziheng Chen, Bernhard Schölkopf, and Nicu Sebe. Hyperbolic Busemann neural networks. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 42029–42038, June 2026.

Christopher Clark, Kenton Lee, Ming-Wei Chang, Tom Kwiatkowski, Michael Collins, and Kristina Toutanova. Boolq: Exploring the surprising difficulty of natural yes/no questions. In *NAACL*, 2019.

Peter Clark, Isaac Cowhey, Oren Etzioni, Tushar Khot, Ashish Sabharwal, Carissa Schoenick, and Oyvind Tafjord. Think you have solved question answering? try arc, the ai2 reasoning challenge. *arXiv preprint arXiv:1803.05457*, 2018.

Karan Desai, Maximilian Nickel, Tanmay Rajpurohit, Justin Johnson, and Shanmukha Ramakrishna Vedantam. Hyperbolic image-text representations. In *International Conference on Machine Learning*, 2023.

Shizhe Diao, Yu Yang, Yonggan Fu, Xin Dong, Dan Su, Markus Kliegl, Zijia Chen, Peter Belcak, Yoshi Suhara, Hongxu Yin, et al. Nemotron-climb: Clustering-based iterative data mixture bootstrapping for language model pre-training. *Advances in Neural Information Processing Systems*, 38, 2026.

Mikhael Gromov. Hyperbolic groups. In *Essays in group theory*, pp. 75–263. Springer, 1987.

Caglar Gulcehre, Misha Denil, Mateusz Malinowski, Ali Razavi, Razvan Pascanu, Karl Moritz Hermann, Peter Battaglia, Victor Bapst, David Raposo, Adam Santoro, et al. Hyperbolic attention networks. In *International Conference on Learning Representations*, 2019.

Yunhui Guo, Xudong Wang, Yubei Chen, and Stella X Yu. Clipped hyperbolic classifiers are super-hyperbolic classifiers. In *2022 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 1–10. IEEE, 2022.

Hulingxiao He, Zhi Tan, and Yuxin Peng. Learning taxonomic trees with hierarchical representation regularization for large multimodal models. *arXiv preprint arXiv:2607.02909*, 2026a.

Neil He, Jiahong Liu, Buze Zhang, Ngoc Bui, Ali Maatouk, Menglin Yang, Irwin King, Melanie Weber, and Rex Ying. Position: Beyond Euclidean–foundation models should embrace Non-Euclidean geometries. *arXiv preprint arXiv:2504.08896*, 2025.

Neil He, Rishabh Anand, Hiren Madhu, Ali Maatouk, Smita Krishnaswamy, Leandros Tassiulas, Menglin Yang, and Rex Ying. Helm: Hyperbolic large language models via mixture-of-curvature experts. *Advances in Neural Information Processing Systems*, 38:142604–142635, 2026b.

Ernst Heintze and Hans-Christoph Im Hof. Geometry of horospheres. *Journal of Differential geometry*, 12(4):481–491, 1977.

Jürgen Jost and Shiping Liu. Ollivier's Ricci curvature, local clustering and curvature-dimension inequalities on graphs. *Discrete & Computational Geometry*, 51(2):300–322, 2014.

Andrej Karpathy. nanochat: The best chatgpt that $100 can buy, 2025. URL `https://github.com/karpathy/nanochat`.

Valentin Khrulkov, Leyla Mirvakhabova, Evgeniya Ustinova, Ivan Oseledets, and Victor Lempitsky. Hyperbolic image embeddings. In *The IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 6418–6428, 2020.

Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

Timo Klein, Thomas Lang, Andrii Shkabrii, Alexander Sturm, Kevin Sidak, Lukas Miklautz, Claudia Plant, Yllka Velaj, and Sebastian Tschiatschek. Understanding and improving hyperbolic deep reinforcement learning. In *International Conference on Learning Representations*, volume 2026, pp. 92200–92235, 2026.

Marc Law, Renjie Liao, Jake Snell, and Richard Zemel. Lorentzian distance learning for hyperbolic representations. In *International conference on machine learning*, pp. 3672–3681. PMLR, 2019.

John M. Lee. *Smooth Manifolds*, pp. 1–29. Springer, 2003.

Hiren Madhu, Ngoc Bui, Ali Maatouk, Leandros Tassiulas, Smita Krishnaswamy, Menglin Yang, Sukanta Ganguly, Kiran Srinivasan, and Rex Ying. Hyprag: Hyperbolic dense retrieval for retrieval augmented generation. In *International Conference on Machine Learning*, 2026.

Todor Mihaylov, Peter Clark, Tushar Khot, and Ashish Sabharwal. Can a suit of armor conduct electricity? a new dataset for open book question answering. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pp. 2381–2391, 2018.

Gal Mishne, Zhengchao Wan, Yusu Wang, and Sheng Yang. The numerical stability of hyperbolic representation learning. In *International Conference on Machine Learning*, pp. 24925–24949. PMLR, 2023.

Maximillian Nickel and Douwe Kiela. Poincaré embeddings for learning hierarchical representations. *Advances in Neural Information Processing Systems*, 30, 2017.

Maximillian Nickel and Douwe Kiela. Learning continuous hierarchies in the Lorentz model of hyperbolic geometry. In *International Conference on Machine Learning*, pp. 3779–3788. PMLR, 2018.

John G Ratcliffe, Sheldon Axler, and Kenneth A Ribet. *Foundations of hyperbolic manifolds*, volume 149. Springer, 1994.

Michael Robinson, Sourya Dey, and Shauna Sweet. The structure of the token space for large language models. *arXiv preprint arXiv:2410.08993*, 2024.

Melissa Roemmele, Cosmin Adrian Bejan, and Andrew S Gordon. Choice of plausible alternatives: An evaluation of commonsense causal reasoning. In *AAAI spring symposium: logical formalizations of commonsense reasoning*, pp. 90–95, 2011.

Frederic Sala, Chris De Sa, Albert Gu, and Christopher Ré. Representation tradeoffs for hyperbolic embeddings. In *International conference on machine learning*, pp. 4460–4469. PMLR, 2018.

Alon Talmor, Jonathan Herzig, Nicholas Lourie, and Jonathan Berant. CommonsenseQA: A question answering challenge targeting commonsense knowledge. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*. Association for Computational Linguistics, 2019.

Philippe Tillet, Hsiang-Tsung Kung, and David Cox. Triton: an intermediate language and compiler for tiled neural network computations. In *Proceedings of the 3rd ACM SIGPLAN International Workshop on Machine Learning and Programming Languages*, 2019.

Canran Xu and Ming Wu. Learning feature interactions with lorentzian factorization machine. In *Proceedings of the AAAI conference on artificial intelligence*, 2020.

Menglin Yang, Harshit Verma, Delvin Ce Zhang, Jiahong Liu, Irwin King, and Rex Ying. Hypformer: Exploring efficient transformer fully in hyperbolic space. In *Proceedings of the 30th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, pp. 3770–3781, 2024.

Menglin Yang, Ram Samarth B B, Aosong Feng, Bo Xiong, Jiahong Liu, Irwin King, and Rex Ying. Hyperbolic fine-tuning for large language models. In *Advances in Neural Information Processing Systems*, 2025.

APPENDIX CONTENTS

A Hyperbolic Space and the Lorentz Model 15

B Busemann Function in the Lorentz Model 16

B.1 Boundary Points of the Lorentz Model 16

B.2 Busemann Function from a General Reference Point 18

B.3 Busemann Function with the Pole as Reference Point 19

C Theoretical Properties of LBL 19

C.1 Geometric Interpretation 20

C.2 Riemannian Gradient of LBL 22

D Curvature Learning Riemannian Adam (CRAdam) 23

D.1 Motivation from the Gradient Geometry of LBL 23

D.2 Fixed-Curvature Riemannian Adam 24

D.3 Riemannian Adam with Learnable Curvature 26

E Training Configurations 30

E.1 Experimental Settings 30

E.2 Model Configurations and Training Budgets 31

F End-to-End Training Efficiency 32

F.1 Forward Computation 33

F.2 Backward Computation and Training Time 34

G Comparison with HELM 35

H Experimental Analysis 37

H.1 Comparison with BMLR 37

H.2 Comparison with Alternative Output Layer Configurations 40

H.3 Riemannian Optimization 41

A HYPERBOLIC SPACE AND THE LORENTZ MODEL

This section provides supplementary background on hyperbolic geometry and the Lorentz model, complementing Section 3.1. Bold italic symbols (e.g., $\boldsymbol{x}$, $\boldsymbol{v}$) denote vectors in the ambient space $\mathbb{R}^{n+1}$, and upright bold symbols (e.g., $\mathbf{u}$, $\mathbf{h}$) denote vectors in $\mathbb{R}^n$; Greek letters are set in bold italic regardless of dimension. For $\boldsymbol{x} \in \mathbb{R}^{n+1}$, we write $\boldsymbol{x} = [x_0, \mathbf{x}_\mathrm{s}]^\top$, where x_0 is the time coordinate and the upright subscript s marks the spatial coordinates; for an expression or an indexed vector, $(\cdot)_\mathrm{s}$ denotes its spatial part, e.g., $(\boldsymbol{v}_j)_\mathrm{s}$. When a coordinate carries further indices, they are ordered as token index, coordinate label, and optimization step, e.g., $x_{j,0}$ for the time coordinate of token j and $x_{0,t}$ for that at update step t (with the token index omitted as in Appendix D). The spatial coordinates of token embeddings are denoted by $\mathbf{w}_j = (\boldsymbol{x}_j)_\mathrm{s}$.

Hyperbolic space is a complete, simply connected Riemannian manifold with constant negative sectional curvature (Lee, 2003). The volume of a geodesic ball in hyperbolic space grows exponentially with its radius, which provides sufficient capacity to embed data whose pairwise relations exhibit a tree-like structure with low distortion (Nickel & Kiela, 2017; Sala et al., 2018). Among its isometric models, the Lorentz model $\mathbb{L}_c^n$ and the Poincaré ball model $\mathbb{P}_c^n$ are the two most widely used (Cannon et al., 1997). We adopt the Lorentz model because it admits closed-form expressions for the geodesic distance and for the exponential and logarithmic maps, and has favorable optimization properties (Mishne et al., 2023). More importantly for this work, the Busemann function $b_{\boldsymbol{\xi}}(\boldsymbol{x})$ in the Lorentz model reduces to the logarithm of a single Lorentz inner product, so that the logits of all tokens can be obtained by one matrix product followed by elementwise operations.

Lorentz Model. The n-dimensional Lorentz model with constant sectional curvature $-c$ ($c > 0$) is the Riemannian manifold $\mathbb{L}_c^n = (\mathcal{L}_c^n, g^{\mathcal{L}})$, whose underlying set is

$$\mathcal{L}_c^n = \left\{ \boldsymbol{x} \in \mathbb{R}^{n+1} : -c\langle \boldsymbol{x}, \boldsymbol{x} \rangle_{\mathcal{L}} = 1,\ x_0 > 0 \right\}, \tag{4}$$

and $g^{\mathcal{L}}$ is the Riemannian metric induced by the Minkowski metric $\mathrm{diag}([-1, 1, \ldots, 1])$ of the ambient space $\mathbb{R}^{n+1}$. Each point is written as $\boldsymbol{x} = [x_0, \mathbf{x}_\mathrm{s}]^\top \in \mathbb{L}_c^n$, where $x_0 \in \mathbb{R}$ is the time coordinate and $\mathbf{x}_\mathrm{s} \in \mathbb{R}^n$ collects the spatial coordinates. The Lorentz inner product is

$$\langle \boldsymbol{x}, \boldsymbol{y} \rangle_{\mathcal{L}} = -x_0 y_0 + \mathbf{x}_\mathrm{s}^\top \mathbf{y}_\mathrm{s}, \tag{5}$$

and the constraint in Eq. 4 determines the time coordinate from the spatial coordinates as $x_0 = \sqrt{1/c + \|\mathbf{x}_\mathrm{s}\|^2}$. Throughout the paper, $\|\cdot\|$ denotes the Euclidean norm, and the Lorentz norm $\|\boldsymbol{v}\|_{\mathcal{L}} = \sqrt{\langle \boldsymbol{v}, \boldsymbol{v} \rangle_{\mathcal{L}}}$ is used only for vectors with $\langle \boldsymbol{v}, \boldsymbol{v} \rangle_{\mathcal{L}} \geq 0$, in particular for tangent vectors.

Tangent Space. For $\boldsymbol{x} \in \mathbb{L}_c^n$, the tangent space $\mathcal{T}_{\boldsymbol{x}}\mathbb{L}_c^n$ is the n-dimensional linear space that locally approximates the manifold at $\boldsymbol{x}$. It consists of all ambient vectors whose Lorentz inner product with $\boldsymbol{x}$ is zero:

$$\mathcal{T}_{\boldsymbol{x}}\mathbb{L}_c^n = \left\{ \boldsymbol{v} \in \mathbb{R}^{n+1} : \langle \boldsymbol{x}, \boldsymbol{v} \rangle_{\mathcal{L}} = 0 \right\}. \tag{6}$$

Although it is embedded in $\mathbb{R}^{n+1}$, $\mathcal{T}_{\boldsymbol{x}}\mathbb{L}_c^n$ has dimension n and is isomorphic to $\mathbb{R}^n$. Moreover, since $\boldsymbol{x}$ satisfies $\langle \boldsymbol{x}, \boldsymbol{x} \rangle_{\mathcal{L}} < 0$, the Lorentz inner product restricted to $\mathcal{T}_{\boldsymbol{x}}\mathbb{L}_c^n$ is positive definite; hence $\|\boldsymbol{v}\|_{\mathcal{L}}$ is well defined for every tangent vector $\boldsymbol{v}$ and vanishes only at $\boldsymbol{v} = \mathbf{0}$.

Distance. For $\boldsymbol{x}, \boldsymbol{y} \in \mathbb{L}_c^n$, the geodesic distance is given by (Ratcliffe et al., 1994; Law et al., 2019)

$$d_{\mathbb{L}}^c(\boldsymbol{x}, \boldsymbol{y}) = \frac{1}{\sqrt{c}} \cosh^{-1}\left(-c\langle \boldsymbol{x}, \boldsymbol{y} \rangle_{\mathcal{L}}\right), \tag{7}$$

where $-c\langle \boldsymbol{x}, \boldsymbol{y} \rangle_{\mathcal{L}} \geq 1$ for all $\boldsymbol{x}, \boldsymbol{y} \in \mathbb{L}_c^n$, with equality if and only if $\boldsymbol{x} = \boldsymbol{y}$.

In the Lorentz model, the squared Lorentz distance has also been used as an efficient distance measure for hyperbolic representation learning (Law et al., 2019; Xu & Wu, 2020; Chen et al., 2022). It avoids the inverse hyperbolic cosine required by the geodesic distance and can be computed directly from the Lorentz inner product, with complexity comparable to the Euclidean dot product. Specifically, it is defined as the squared Lorentzian norm of the difference between two points in the ambient Minkowski space:

$$d_c^2(\boldsymbol{x}, \boldsymbol{y}) = \|\boldsymbol{x} - \boldsymbol{y}\|_{\mathcal{L}}^2 = -\frac{2}{c} - 2\langle \boldsymbol{x}, \boldsymbol{y} \rangle_{\mathcal{L}}. \tag{8}$$

Note that $d_c^2(\boldsymbol{x}, \boldsymbol{y}) \neq \left(d_{\mathbb{L}}^c(\boldsymbol{x}, \boldsymbol{y})\right)^2$ for distinct points $\boldsymbol{x} \neq \boldsymbol{y}$; the squared Lorentz distance is not the squared geodesic distance.

Exponential and Logarithmic Maps. The exponential and logarithmic maps relate points on $\mathbb{L}_c^n$ to vectors in its tangent spaces. For $\boldsymbol{x} \in \mathbb{L}_c^n$ and $\boldsymbol{v} \in \mathcal{T}_{\boldsymbol{x}}\mathbb{L}_c^n$, the exponential map $\exp_{\boldsymbol{x}}^c(\boldsymbol{v})$ returns the point reached at unit time by the geodesic that starts at $\boldsymbol{x}$ with initial velocity $\boldsymbol{v}$. For $\boldsymbol{v} \neq \mathbf{0}$, it is given by (Nickel & Kiela, 2018):

$$\exp_{\boldsymbol{x}}^c(\boldsymbol{v}) = \cosh\left(\sqrt{c}\|\boldsymbol{v}\|_{\mathcal{L}}\right)\boldsymbol{x} + \frac{1}{\sqrt{c}}\sinh\left(\sqrt{c}\|\boldsymbol{v}\|_{\mathcal{L}}\right)\frac{\boldsymbol{v}}{\|\boldsymbol{v}\|_{\mathcal{L}}}, \tag{9}$$

and $\exp_{\boldsymbol{x}}^c(\mathbf{0}) = \boldsymbol{x}$ by continuity. Conversely, for $\boldsymbol{y} \in \mathbb{L}_c^n$ with $\boldsymbol{y} \neq \boldsymbol{x}$, the logarithmic map $\log_{\boldsymbol{x}}^c(\boldsymbol{y})$ returns the tangent vector at $\boldsymbol{x}$ that points along the unique geodesic from $\boldsymbol{x}$ to $\boldsymbol{y}$ and whose Lorentz norm equals $d_{\mathbb{L}}^c(\boldsymbol{x}, \boldsymbol{y})$:

$$\log_{\boldsymbol{x}}^c(\boldsymbol{y}) = d_{\mathbb{L}}^c(\boldsymbol{x}, \boldsymbol{y})\frac{\boldsymbol{y} + c\langle \boldsymbol{x}, \boldsymbol{y}\rangle_{\mathcal{L}}\boldsymbol{x}}{\|\boldsymbol{y} + c\langle \boldsymbol{x}, \boldsymbol{y}\rangle_{\mathcal{L}}\boldsymbol{x}\|_{\mathcal{L}}}, \tag{10}$$

and $\log_{\boldsymbol{x}}^c(\boldsymbol{x}) = \mathbf{0}$.

Since $\mathbb{L}_c^n$ is complete and simply connected with negative curvature, $\exp_{\boldsymbol{x}}^c$ is a diffeomorphism from $\mathcal{T}_{\boldsymbol{x}}\mathbb{L}_c^n$ onto $\mathbb{L}_c^n$, and $\log_{\boldsymbol{x}}^c$ is its inverse.

We use the pole

$$\boldsymbol{o} = \left[\frac{1}{\sqrt{c}}, \mathbf{0}\right]^\top \in \mathbb{L}_c^n, \tag{11}$$

which serves both as the base point at which hidden states are lifted to the tangent space (Section 4.1) and as the reference point of the Busemann function $b_{\boldsymbol{\xi}}(\boldsymbol{x})$. Since $\langle \boldsymbol{o}, \boldsymbol{v}\rangle_{\mathcal{L}} = -v_0/\sqrt{c}$ for any $\boldsymbol{v} = [v_0, \mathbf{v}]^\top \in \mathbb{R}^{n+1}$, the tangent space at the pole is

$$\mathcal{T}_{\boldsymbol{o}}\mathbb{L}_c^n = \left\{[0, \mathbf{v}]^\top : \mathbf{v} \in \mathbb{R}^n\right\}, \tag{12}$$

and $\|[0, \mathbf{v}]^\top\|_{\mathcal{L}} = \|\mathbf{v}\|$. Substituting into Eq. 9, the exponential map at the pole is, for $\mathbf{v} \neq \mathbf{0}$,

$$\exp_{\boldsymbol{o}}^c\left([0, \mathbf{v}]^\top\right) = \begin{bmatrix} \frac{1}{\sqrt{c}}\cosh\left(\sqrt{c}\|\mathbf{v}\|\right) \\ \frac{1}{\sqrt{c}}\sinh\left(\sqrt{c}\|\mathbf{v}\|\right)\frac{\mathbf{v}}{\|\mathbf{v}\|} \end{bmatrix}, \tag{13}$$

with $\exp_{\boldsymbol{o}}^c(\mathbf{0}) = \boldsymbol{o}$. Setting $\mathbf{v} = \mathbf{h}$ recovers Eq. 1.

Conversely, let $\boldsymbol{x} = [x_0, \mathbf{x}_{\mathrm{s}}]^\top \in \mathbb{L}_c^n$ with $\boldsymbol{x} \neq \boldsymbol{o}$, or equivalently $\mathbf{x}_{\mathrm{s}} \neq \mathbf{0}$. Then $\boldsymbol{x} + c\langle \boldsymbol{o}, \boldsymbol{x}\rangle_{\mathcal{L}}\boldsymbol{o} = [0, \mathbf{x}_{\mathrm{s}}]^\top$ and $d_{\mathbb{L}}^c(\boldsymbol{o}, \boldsymbol{x}) = \frac{1}{\sqrt{c}}\cosh^{-1}(\sqrt{c}\,x_0)$, so Eq. 10 gives

$$\log_{\boldsymbol{o}}^c(\boldsymbol{x}) = \begin{bmatrix} 0 \\ \frac{\cosh^{-1}\left(\sqrt{c}\,x_0\right)}{\sqrt{c}}\frac{\mathbf{x}_{\mathrm{s}}}{\|\mathbf{x}_{\mathrm{s}}\|} \end{bmatrix}, \tag{14}$$

with $\log_{\boldsymbol{o}}^c(\boldsymbol{o}) = \mathbf{0}$.

B Busemann Function in the Lorentz Model

This section derives the closed form Busemann function $b_{\boldsymbol{\xi}}(\boldsymbol{x})$. Since a Busemann function is determined by a geodesic ray, and hence by both its endpoint at infinity and its starting point, we proceed from the general case to the one adopted in this paper. Appendix B.1 characterizes the points at infinity of $\mathbb{L}_c^n$ and their normalized representation. Appendix B.2 derives the Busemann function of the geodesic ray from an arbitrary reference point $\boldsymbol{p} \in \mathbb{L}_c^n$ toward a boundary point. Appendix B.3 specializes the result to the pole $\boldsymbol{o}$, which serves as the reference point throughout the paper.

B.1 Boundary Points of the Lorentz Model

Asymptotic direction of a geodesic ray. Let $\boldsymbol{p} \in \mathbb{L}_c^n$ and let $\boldsymbol{v} \in \mathcal{T}_{\boldsymbol{p}}\mathbb{L}_c^n$ with $\|\boldsymbol{v}\|_{\mathcal{L}} = 1$. By Eq. 9, the unit-speed geodesic ray starting from $\boldsymbol{p}$ with initial velocity $\boldsymbol{v}$ is

$$\gamma(t) = \exp_{\boldsymbol{p}}^c(t\boldsymbol{v}) = \cosh(\sqrt{c}\,t)\,\boldsymbol{p} + \frac{1}{\sqrt{c}}\sinh(\sqrt{c}\,t)\,\boldsymbol{v}, \qquad t \geq 0. \tag{15}$$

Its coordinates diverge as $t \to \infty$, so γ has no limit on $\mathbb{L}_c^n$. Its direction in the ambient space, however, converges: since $\cosh(\sqrt{c}\,t) \sim \sinh(\sqrt{c}\,t) \sim \frac{1}{2}e^{\sqrt{c}\,t}$,

$$\lim_{t\to\infty} 2e^{-\sqrt{c}\,t}\,\gamma(t) = \boldsymbol{p} + \frac{1}{\sqrt{c}}\boldsymbol{v} =: \boldsymbol{\zeta}. \tag{16}$$

The limit $\boldsymbol{\zeta}$ is a null vector, since $\langle \boldsymbol{\zeta}, \boldsymbol{\zeta}\rangle_{\mathcal{L}} = \langle \boldsymbol{p}, \boldsymbol{p}\rangle_{\mathcal{L}} + \frac{2}{\sqrt{c}}\langle \boldsymbol{p}, \boldsymbol{v}\rangle_{\mathcal{L}} + \frac{1}{c}\langle \boldsymbol{v}, \boldsymbol{v}\rangle_{\mathcal{L}} = -\frac{1}{c} + 0 + \frac{1}{c} = 0$. It is nonzero because $\langle \boldsymbol{p}, \boldsymbol{\zeta}\rangle_{\mathcal{L}} = -1/c$, and its time coordinate is positive: it is nonnegative as a limit of positive multiples of points with positive time coordinates, and it cannot vanish because a nonzero null vector satisfies $|\zeta_0| = \|\boldsymbol{\zeta}_{\mathrm{s}}\| > 0$.

Boundary points. We call $\boldsymbol{\xi} = [\xi_0, \boldsymbol{\xi}_{\mathrm{s}}]^\top \in \mathbb{R}^{n+1}$ a future-pointing null vector if $\langle \boldsymbol{\xi}, \boldsymbol{\xi}\rangle_{\mathcal{L}} = 0$ and $\xi_0 > 0$. The boundary at infinity $\partial_\infty \mathbb{L}_c^n$ is identified with the set of future-pointing null vectors modulo positive rescaling, $\boldsymbol{\xi} \sim \lambda\boldsymbol{\xi}$ for $\lambda > 0$: two geodesic rays have the same endpoint at infinity if and only if their limiting vectors in Eq. 16 are positive multiples of each other (Ratcliffe et al., 1994). We therefore refer to null vector $\boldsymbol{\xi}$ as a representative of the boundary point it determines, and, following the main text, call such a representative a boundary direction.

Normalized representative. For a future-pointing null vector, $\langle \boldsymbol{\xi}, \boldsymbol{\xi}\rangle_{\mathcal{L}} = -\xi_0^2 + \|\boldsymbol{\xi}_{\mathrm{s}}\|^2 = 0$ gives $\|\boldsymbol{\xi}_{\mathrm{s}}\| = \xi_0 > 0$. Dividing by ξ_0 yields the equivalent representative

$$\frac{\boldsymbol{\xi}}{\xi_0} = \begin{bmatrix} 1 \\ \mathbf{u} \end{bmatrix}, \qquad \mathbf{u} = \frac{\boldsymbol{\xi}_{\mathrm{s}}}{\xi_0}, \qquad \|\mathbf{u}\| = 1. \tag{17}$$

Hence every boundary point has a unique representative of the form $[1, \mathbf{u}]^\top$ with a unit vector $\mathbf{u} \in \mathbb{R}^n$, which we call the normalized representative. For the pole $\boldsymbol{o} = [1/\sqrt{c}, \mathbf{0}]^\top$ and $\bar{\boldsymbol{u}} = [0, \mathbf{u}]^\top \in \mathcal{T}_{\boldsymbol{o}}\mathbb{L}_c^n$, which satisfies $\|\bar{\boldsymbol{u}}\|_{\mathcal{L}} = 1$, Eq. 15 gives the ray

$$\gamma_{\mathbf{u}}(t) = \exp_{\boldsymbol{o}}^c(t\bar{\boldsymbol{u}}) = \frac{1}{\sqrt{c}}\begin{bmatrix} \cosh(\sqrt{c}\,t) \\ \sinh(\sqrt{c}\,t)\,\mathbf{u} \end{bmatrix}, \qquad t \geq 0, \tag{18}$$

whose limiting vector is $\boldsymbol{\zeta} = \frac{1}{\sqrt{c}}[1, \mathbf{u}]^\top$. Its endpoint is therefore the boundary point with normalized representative $[1, \mathbf{u}]^\top$, so every unit vector $\mathbf{u}$ arises as the endpoint of a unique ray emanating from $\boldsymbol{o}$. In LBL we take $\mathbf{u} = \mathbf{h}/\|\mathbf{h}\|$ for a nonzero hidden state $\mathbf{h}$, which gives the boundary direction $\boldsymbol{\xi}_{\mathbf{h}} = [1, \mathbf{u}]^\top$ used in Section 4.1. While the boundary point itself is independent of the choice of representative, the closed form Busemann function $b_{\boldsymbol{\xi}}(\boldsymbol{x})$ is not, as it leaves the reference point implicit; Appendix B.3 shows that the normalized representative amounts to choosing $\boldsymbol{o}$ as the reference point.

Sign of the inner product with a boundary representative. The following lemma ensures that all logarithms below are well defined.

Lemma 1. *Let $\boldsymbol{x} \in \mathbb{L}_c^n$ and let $\boldsymbol{\xi}$ be a future-pointing null vector. Then $\langle \boldsymbol{x}, \boldsymbol{\xi}\rangle_{\mathcal{L}} < 0$.*

Proof. By the Cauchy–Schwarz inequality and $\|\boldsymbol{\xi}_{\mathrm{s}}\| = \xi_0$,

$$\begin{aligned} \langle \boldsymbol{x}, \boldsymbol{\xi}\rangle_{\mathcal{L}} &= -x_0\xi_0 + \mathbf{x}_{\mathrm{s}}^\top \boldsymbol{\xi}_{\mathrm{s}} \\ &\leq -x_0\xi_0 + \|\mathbf{x}_{\mathrm{s}}\|\,\xi_0 \\ &= \xi_0\big(\|\mathbf{x}_{\mathrm{s}}\| - x_0\big) < 0, \end{aligned} \tag{19}$$

where the last inequality uses $\xi_0 > 0$ and $x_0 = \sqrt{1/c + \|\mathbf{x}_{\mathrm{s}}\|^2} > \|\mathbf{x}_{\mathrm{s}}\|$. □

Applied to the boundary direction determined by a hidden state, Lemma 1 gives the well-definedness of LBL.

Corollary 2 (Well-Definedness of the LBL)**.** *Let $\boldsymbol{x}_j = [x_{j,0}, \mathbf{w}_j]^\top \in \mathbb{L}_c^n$ with $c > 0$, and let $\mathbf{u} \in \mathbb{R}^n$ satisfy $\|\mathbf{u}\| = 1$. Then $x_{j,0} - \mathbf{w}_j^\top \mathbf{u} > 0$. Consequently, the argument of the logarithm in Eq. 2, $-\sqrt{c}\,\langle \boldsymbol{x}_j, \boldsymbol{\xi}_{\mathbf{h}}\rangle_{\mathcal{L}} = \sqrt{c}\,(x_{j,0} - \mathbf{w}_j^\top \mathbf{u})$, is strictly positive, and $\ell_j(\mathbf{h})$ is well defined for every $\boldsymbol{x}_j \in \mathbb{L}_c^n$ and every nonzero hidden state $\mathbf{h}$.*

Proof. The boundary direction $\boldsymbol{\xi}_{\mathbf{h}} = [1, \mathbf{u}]^\top$ satisfies $\langle \boldsymbol{\xi}_{\mathbf{h}}, \boldsymbol{\xi}_{\mathbf{h}}\rangle_{\mathcal{L}} = -1 + \|\mathbf{u}\|^2 = 0$ and has time coordinate $1 > 0$, so it is a future-pointing null vector. Lemma 1 gives $\langle \boldsymbol{x}_j, \boldsymbol{\xi}_{\mathbf{h}}\rangle_{\mathcal{L}} = -x_{j,0} + \mathbf{w}_j^\top \mathbf{u} < 0$, and multiplying by $-\sqrt{c} < 0$ yields the claim. □

B.2 Busemann Function from a General Reference Point

Let $\boldsymbol{p} \in \mathbb{L}_c^n$ be a reference point and let $\boldsymbol{\xi}$ be a representative of a boundary point.

Geodesic ray from $\boldsymbol{p}$ toward $\boldsymbol{\xi}$. By Lemma 1, $\langle \boldsymbol{p}, \boldsymbol{\xi} \rangle_{\mathcal{L}} < 0$, so we define

$$\mu = -\frac{1}{c\langle \boldsymbol{p}, \boldsymbol{\xi} \rangle_{\mathcal{L}}} > 0, \qquad \boldsymbol{v} = \sqrt{c}\,(\mu\boldsymbol{\xi} - \boldsymbol{p}). \tag{20}$$

Using $\langle \boldsymbol{p}, \boldsymbol{p} \rangle_{\mathcal{L}} = -1/c$, $\langle \boldsymbol{\xi}, \boldsymbol{\xi} \rangle_{\mathcal{L}} = 0$, and $\mu\langle \boldsymbol{p}, \boldsymbol{\xi} \rangle_{\mathcal{L}} = -1/c$, we have

$$\langle \boldsymbol{v}, \boldsymbol{p} \rangle_{\mathcal{L}} = \sqrt{c}\Big(-\frac{1}{c} + \frac{1}{c} \Big) = 0, \tag{21}$$

$$\langle \boldsymbol{v}, \boldsymbol{v} \rangle_{\mathcal{L}} = c\Big(0 + \frac{2}{c} - \frac{1}{c} \Big) = 1, \tag{22}$$

so $\boldsymbol{v}$ is a unit tangent vector at $\boldsymbol{p}$. Let $\gamma_{\boldsymbol{p},\boldsymbol{\xi}}(t) = \exp_{\boldsymbol{p}}^c(t\boldsymbol{v})$ denote the corresponding geodesic ray, given by Eq. 15. Its limiting vector is $\boldsymbol{p} + \boldsymbol{v}/\sqrt{c} = \mu\boldsymbol{\xi}$, so its endpoint is the boundary point represented by $\boldsymbol{\xi}$. This ray is unique: if a unit tangent vector $\boldsymbol{v}'$ at $\boldsymbol{p}$ satisfies $\boldsymbol{p} + \boldsymbol{v}'/\sqrt{c} = \mu'\boldsymbol{\xi}$ for some $\mu' > 0$, taking the Lorentz inner product with $\boldsymbol{p}$ gives $\mu'\langle \boldsymbol{p}, \boldsymbol{\xi} \rangle_{\mathcal{L}} = -1/c$, hence $\mu' = \mu$ and $\boldsymbol{v}' = \boldsymbol{v}$.

Closed form. The Busemann function of $\gamma_{\boldsymbol{p},\boldsymbol{\xi}}$ is

$$b_{\boldsymbol{\xi}}^{\boldsymbol{p}}(\boldsymbol{x}) = \lim_{t\to\infty} \Big[d_{\mathbb{L}}^c\big(\boldsymbol{x}, \gamma_{\boldsymbol{p},\boldsymbol{\xi}}(t)\big) - t \Big], \qquad \boldsymbol{x} \in \mathbb{L}_c^n. \tag{23}$$

Let $\rho = \sqrt{c}\,t$, $A = -c\langle \boldsymbol{x}, \boldsymbol{p} \rangle_{\mathcal{L}}$, and $B = -\sqrt{c}\langle \boldsymbol{x}, \boldsymbol{v} \rangle_{\mathcal{L}}$. Substituting Eq. 15 and using $\cosh\rho = (e^{\rho} + e^{-\rho})/2$ and $\sinh\rho = (e^{\rho} - e^{-\rho})/2$, we obtain

$$\begin{aligned} -c\big\langle \boldsymbol{x}, \gamma_{\boldsymbol{p},\boldsymbol{\xi}}(t) \big\rangle_{\mathcal{L}} &= A\cosh\rho + B\sinh\rho \\ &= \frac{1}{2}\Big[(A+B)e^{\rho} + (A-B)e^{-\rho} \Big]. \end{aligned} \tag{24}$$

Since $\boldsymbol{p} + \boldsymbol{v}/\sqrt{c} = \mu\boldsymbol{\xi}$, the leading coefficient is

$$\begin{aligned} A + B &= -c\Big\langle \boldsymbol{x}, \boldsymbol{p} + \frac{1}{\sqrt{c}}\boldsymbol{v} \Big\rangle_{\mathcal{L}} \\ &= -c\mu\langle \boldsymbol{x}, \boldsymbol{\xi} \rangle_{\mathcal{L}} \\ &= \frac{\langle \boldsymbol{x}, \boldsymbol{\xi} \rangle_{\mathcal{L}}}{\langle \boldsymbol{p}, \boldsymbol{\xi} \rangle_{\mathcal{L}}}, \end{aligned} \tag{25}$$

which is strictly positive by Lemma 1. Hence, as $\rho \to \infty$,

$$-c\big\langle \boldsymbol{x}, \gamma_{\boldsymbol{p},\boldsymbol{\xi}}(t) \big\rangle_{\mathcal{L}} = \frac{1}{2}\,\frac{\langle \boldsymbol{x}, \boldsymbol{\xi} \rangle_{\mathcal{L}}}{\langle \boldsymbol{p}, \boldsymbol{\xi} \rangle_{\mathcal{L}}}\, e^{\rho}\big(1 + o(1)\big). \tag{26}$$

Using $\cosh^{-1}(z) = \log\big(z + \sqrt{z^2 - 1}\big) = \log(2z) + o(1)$ as $z \to \infty$, we have

$$\cosh^{-1}\Big(-c\big\langle \boldsymbol{x}, \gamma_{\boldsymbol{p},\boldsymbol{\xi}}(t) \big\rangle_{\mathcal{L}} \Big) = \rho + \log\frac{\langle \boldsymbol{x}, \boldsymbol{\xi} \rangle_{\mathcal{L}}}{\langle \boldsymbol{p}, \boldsymbol{\xi} \rangle_{\mathcal{L}}} + o(1). \tag{27}$$

Dividing by $\sqrt{c}$, subtracting $t = \rho/\sqrt{c}$, and letting $t \to \infty$ yields

$$b_{\boldsymbol{\xi}}^{\boldsymbol{p}}(\boldsymbol{x}) = \frac{1}{\sqrt{c}}\log\frac{\langle \boldsymbol{x}, \boldsymbol{\xi} \rangle_{\mathcal{L}}}{\langle \boldsymbol{p}, \boldsymbol{\xi} \rangle_{\mathcal{L}}}. \tag{28}$$

Properties. Eq. 28 has the following properties.

(i) Independence of the representative. Replacing $\boldsymbol{\xi}$ by $\lambda\boldsymbol{\xi}$ with $\lambda > 0$ multiplies the numerator and the denominator by λ, so $b_{\boldsymbol{\xi}}^{\boldsymbol{p}}$ depends on $\boldsymbol{\xi}$ only through the boundary point it represents.

(ii) Normalization at the reference point. $b_{\boldsymbol{\xi}}^{\boldsymbol{p}}(\boldsymbol{p}) = 0$.

(iii) Change of reference point. For any $\boldsymbol{p}, \boldsymbol{p}' \in \mathbb{L}_c^n$,

$$b_{\boldsymbol{\xi}}^{\boldsymbol{p}'}(\boldsymbol{x}) = b_{\boldsymbol{\xi}}^{\boldsymbol{p}}(\boldsymbol{x}) - b_{\boldsymbol{\xi}}^{\boldsymbol{p}}(\boldsymbol{p}') \qquad \text{for all } \boldsymbol{x} \in \mathbb{L}_c^n. \tag{29}$$

Thus, Busemann functions associated with the same boundary point differ only by an additive constant, in agreement with the general property of Busemann functions on spaces of non-positive curvature (Bridson & Haefliger, 2013).

(iv) Sign convention. Since $\langle \boldsymbol{v}, \boldsymbol{\xi} \rangle_{\mathcal{L}} = -\sqrt{c}\,\langle \boldsymbol{p}, \boldsymbol{\xi} \rangle_{\mathcal{L}}$, Eq. 15 gives $\langle \gamma_{\boldsymbol{p},\boldsymbol{\xi}}(s), \boldsymbol{\xi} \rangle_{\mathcal{L}} = e^{-\sqrt{c}\,s} \langle \boldsymbol{p}, \boldsymbol{\xi} \rangle_{\mathcal{L}}$, and therefore $b_{\boldsymbol{\xi}}^{\boldsymbol{p}}(\gamma_{\boldsymbol{p},\boldsymbol{\xi}}(s)) = -s$ for all $s \geq 0$. The Busemann function decreases linearly along the ray toward the boundary point.

(v) Horospheres. The level set $\{\boldsymbol{x} \in \mathbb{L}_c^n : b_{\boldsymbol{\xi}}^{\boldsymbol{p}}(\boldsymbol{x}) = \tau\}$ equals $\{\boldsymbol{x} \in \mathbb{L}_c^n : \langle \boldsymbol{x}, \boldsymbol{\xi} \rangle_{\mathcal{L}} = e^{\sqrt{c}\,\tau} \langle \boldsymbol{p}, \boldsymbol{\xi} \rangle_{\mathcal{L}}\}$, i.e., the intersection of $\mathbb{L}_c^n$ with an affine hyperplane in $\mathbb{R}^{n+1}$ on which the Lorentz inner product with $\boldsymbol{\xi}$ is constant. By (iii), replacing $\boldsymbol{p}$ with $\boldsymbol{p}'$ shifts every level τ by the same constant $b_{\boldsymbol{\xi}}^{\boldsymbol{p}}(\boldsymbol{p}')$. The family of horospheres is therefore determined by the boundary point alone, and the reference point only relabels the levels τ.

B.3 Busemann Function with the Pole as Reference Point

Throughout this paper, we take the pole $\boldsymbol{o} = [1/\sqrt{c}, \mathbf{0}]^\top$ as the reference point. This choice is consistent with the construction of LBL: hidden states are lifted to $\mathcal{T}_{\boldsymbol{o}}\mathbb{L}_c^n$, so the geodesic ray $\gamma_{\mathbf{u}}$ associated with a hidden state starts at $\boldsymbol{o}$ (Eq. 1). We write $b_{\boldsymbol{\xi}} = b_{\boldsymbol{\xi}}^{\boldsymbol{o}}$ and use the normalized representative $\boldsymbol{\xi} = [1, \mathbf{u}]^\top$ with $\|\mathbf{u}\| = 1$.

Closed form. Since $\langle \boldsymbol{o}, \boldsymbol{\xi} \rangle_{\mathcal{L}} = -1/\sqrt{c}$, Eq. 28 becomes

$$\begin{aligned} b_{\boldsymbol{\xi}}(\boldsymbol{x}) &= \frac{1}{\sqrt{c}} \log\left(-\sqrt{c}\,\langle \boldsymbol{x}, \boldsymbol{\xi} \rangle_{\mathcal{L}}\right) \\ &= \frac{1}{\sqrt{c}} \log\left[\sqrt{c}\left(x_0 - \mathbf{x}_{\mathrm{s}}^\top \mathbf{u}\right)\right], \end{aligned} \tag{30}$$

which is the Busemann function in the main text. In this case, Eq. 20 gives $\mu = 1/\sqrt{c}$ and $\boldsymbol{v} = [0, \mathbf{u}]^\top$, so the ray $\gamma_{\boldsymbol{o},\boldsymbol{\xi}}$ coincides with $\gamma_{\mathbf{u}}$ in Eq. 18. By properties (ii) and (iv), $b_{\boldsymbol{\xi}}(\boldsymbol{o}) = 0$ and $b_{\boldsymbol{\xi}}(\gamma_{\mathbf{u}}(s)) = -s$; explicitly, $-\sqrt{c}\,\langle \gamma_{\mathbf{u}}(s), \boldsymbol{\xi} \rangle_{\mathcal{L}} = \cosh(\sqrt{c}\,s) - \sinh(\sqrt{c}\,s) = e^{-\sqrt{c}\,s}$. Hence $b_{\boldsymbol{\xi}}$ decreases linearly along $\gamma_{\mathbf{u}}$, whereas $-b_{\boldsymbol{\xi}}(\boldsymbol{x})$ increases toward the boundary point.

Role of the normalization $\xi_0 = 1$. Unlike Eq. 28, the closed form in Eq. 30 does not display its reference point and is not invariant to rescaling the representative: for $\lambda > 0$,

$$\frac{1}{\sqrt{c}} \log\left(-\sqrt{c}\,\langle \boldsymbol{x}, \lambda\boldsymbol{\xi} \rangle_{\mathcal{L}}\right) = b_{\boldsymbol{\xi}}(\boldsymbol{x}) + \frac{1}{\sqrt{c}} \log \lambda. \tag{31}$$

By Eq. 29, the right-hand side equals $b_{\boldsymbol{\xi}}^{\boldsymbol{p}}(\boldsymbol{x})$ for any reference point $\boldsymbol{p}$ with $b_{\boldsymbol{\xi}}(\boldsymbol{p}) = -\frac{1}{\sqrt{c}} \log \lambda$. The reference point of the closed form is thus encoded in the scale of the representative, and $\langle \boldsymbol{o}, \boldsymbol{\xi} \rangle_{\mathcal{L}} = -1/\sqrt{c}$, i.e., $\xi_0 = 1$, is exactly the normalization under which it corresponds to $\boldsymbol{o}$. The normalized representative is therefore not merely a convenient scale: it fixes the additive constant of the Busemann function through $b_{\boldsymbol{\xi}}(\boldsymbol{o}) = 0$.

Implication for LBL. Replacing the reference point $\boldsymbol{o}$ by any $\boldsymbol{p} \in \mathbb{L}_c^n$, or rescaling $\boldsymbol{\xi}_{\mathbf{h}}$ by $\lambda > 0$, changes the LBL in Eq. 2 to $\ell_j(\mathbf{h}) + \alpha\, b_{\boldsymbol{\xi}_{\mathbf{h}}}(\boldsymbol{p})$, with the same shift for every token $j \in \mathcal{V}$. Consequently, the softmax distribution, the cross entropy loss, and the logit differences $\ell_j(\mathbf{h}) - \ell_k(\mathbf{h})$ do not depend on the reference point. In particular, since $\boldsymbol{z}_{\mathbf{h}} = \gamma_{\mathbf{u}}(\|\mathbf{h}\|)$ satisfies $b_{\boldsymbol{\xi}_{\mathbf{h}}}(\boldsymbol{z}_{\mathbf{h}}) = -\|\mathbf{h}\|$, taking $\boldsymbol{z}_{\mathbf{h}}$ as the reference point shifts every logit by $-\alpha\|\mathbf{h}\|$ and leaves the distribution unchanged.

C Theoretical Properties of LBL

This section studies LBL as a function of the token embedding for a fixed hidden state; its well-definedness is established in Corollary 2. Appendix C.1 interprets its values along geodesics and in terms of the distance from the pole, and Appendix C.2 characterizes its Riemannian gradient.

C.1 GEOMETRIC INTERPRETATION

We interpret LBL in two steps. Proposition 3 considers token embeddings on the geodesic through the pole $\boldsymbol{o}$ in the context direction $\mathbf{u}$, where LBL reduces to a signed distance from $\boldsymbol{o}$. Proposition 4 then treats an arbitrary token embedding through its distance from $\boldsymbol{o}$ and its angle to $\mathbf{u}$.

Proposition 3 (Geodesic Interpretation of LBL)**.** *Let $\boldsymbol{\xi}_{\mathbf{h}} = [1, \mathbf{u}]^\top$ be the boundary direction determined by the unit spatial direction $\mathbf{u}$, and let $t \geq 0$. If a token embedding lies on the geodesic ray from $\boldsymbol{o}$ toward $\boldsymbol{\xi}_{\mathbf{h}}$,*

$$\boldsymbol{x}_j = \gamma_{\mathbf{u}}(t), \qquad \textit{then} \qquad \ell_j(\mathbf{h}) = -\alpha\, b_{\boldsymbol{\xi}_{\mathbf{h}}}(\boldsymbol{x}_j) = \alpha t. \tag{32}$$

If it lies on the opposite ray, $\boldsymbol{x}_j = \gamma_{-\mathbf{u}}(t)$, then $\ell_j(\mathbf{h}) = -\alpha t$.

Proof. By Eq. 18,

$$\gamma_{\pm\mathbf{u}}(t) = \frac{1}{\sqrt{c}} \begin{bmatrix} \cosh(\sqrt{c}\, t) \\ \pm \sinh(\sqrt{c}\, t)\, \mathbf{u} \end{bmatrix}. \tag{33}$$

Since $\|\mathbf{u}\| = 1$, the argument of the logarithm in Eq. 2 is

$$\sqrt{c}\left(x_{j,0} - \mathbf{w}_j^\top \mathbf{u}\right) = \cosh(\sqrt{c}\, t) \mp \sinh(\sqrt{c}\, t) = e^{\mp\sqrt{c}\, t}, \tag{34}$$

and therefore $\ell_j(\mathbf{h}) = -\frac{\alpha}{\sqrt{c}} \log\left(e^{\mp\sqrt{c}\, t}\right) = \pm\alpha t$. □

The two rays $\gamma_{\mathbf{u}}$ and $\gamma_{-\mathbf{u}}$ together form the geodesic through $\boldsymbol{o}$ in the direction $\mathbf{u}$. Proposition 3 shows that, along this geodesic, LBL equals α times the signed distance from $\boldsymbol{o}$: it is positive and grows linearly toward $\boldsymbol{\xi}_{\mathbf{h}}$, and it is negative and decreases linearly in the opposite direction. This agrees with $b_{\boldsymbol{\xi}}(\gamma_{\mathbf{u}}(s)) = -s$ in Appendix B.3 and makes precise the interpretation of $-b_{\boldsymbol{\xi}}$ as a coordinate along $\boldsymbol{\xi}$ in Section 3.2.

Dependence on the reference point. The values in Proposition 3 are measured relative to the reference point $\boldsymbol{o}$: t is the distance from $\boldsymbol{o}$, and, since $b_{\boldsymbol{\xi}_{\mathbf{h}}}(\boldsymbol{o}) = 0$, $\ell_j(\mathbf{h}) > 0$ holds if and only if $b_{\boldsymbol{\xi}_{\mathbf{h}}}(\boldsymbol{x}_j) < b_{\boldsymbol{\xi}_{\mathbf{h}}}(\boldsymbol{o})$, i.e., $\boldsymbol{x}_j$ lies on the side of the horosphere $\mathcal{H}_{\boldsymbol{\xi}_{\mathbf{h}},0}$, which passes through $\boldsymbol{o}$, facing $\boldsymbol{\xi}_{\mathbf{h}}$. Equivalently, a positive logit means that token j is ranked above a hypothetical token placed at the pole, whose logit is zero. The same holds for a Euclidean output layer without a bias term, where a token at the origin has logit zero, so the signs of logits in the two output layers can be compared. As shown in Appendix B.3, logit differences, and hence the next-token distribution, do not depend on this choice.

General token embeddings. Proposition 3 covers only embeddings on the geodesic through $\boldsymbol{o}$ in the direction $\mathbf{u}$. A general embedding $\boldsymbol{x}_j = [x_{j,0}, \mathbf{w}_j]^\top$ with $\mathbf{w}_j \neq \mathbf{0}$ is described by its distance d_j from the pole and the angle θ_j between $\mathbf{w}_j$ and $\mathbf{u}$; the embeddings in Proposition 3 are those with $\theta_j = 0$ and $\theta_j = \pi$, for which $d_j = t$.

Proposition 4 (Radial Specificity of LBL)**.** *Let $\boldsymbol{x}_j = [x_{j,0}, \mathbf{w}_j]^\top \in \mathbb{L}_c^n$ with $\mathbf{w}_j \neq \mathbf{0}$, let $d_j = d_{\mathbb{L}}^c(\boldsymbol{o}, \boldsymbol{x}_j)$, and let θ_j denote the angle between $\mathbf{w}_j$ and $\mathbf{u} = \mathbf{h}/\|\mathbf{h}\|$. Then*

$$\ell_j(\mathbf{h}) = -\frac{\alpha}{\sqrt{c}} \log\left(\cosh(\sqrt{c}\, d_j) - \sinh(\sqrt{c}\, d_j)\cos\theta_j\right). \tag{35}$$

Consequently:

- *(i) $-\alpha d_j \leq \ell_j(\mathbf{h}) \leq \alpha d_j$, where the upper and lower bounds are attained at $\theta_j = 0$ and $\theta_j = \pi$, respectively, i.e., for the embeddings $\gamma_{\mathbf{u}}(d_j)$ and $\gamma_{-\mathbf{u}}(d_j)$ in Proposition 3;*
- *(ii) $\ell_j(\mathbf{h}) > 0$ if and only if $\cos\theta_j > \tanh(\sqrt{c}\, d_j/2)$;*
- *(iii) at $\theta_j = \pi/2$, $\ell_j(\mathbf{h}) = -\frac{\alpha}{\sqrt{c}} \log\cosh(\sqrt{c}\, d_j) < 0$.*

Proof. Expression. Since $d_j = \frac{1}{\sqrt{c}} \cosh^{-1}(\sqrt{c}\, x_{j,0})$ and $x_{j,0}^2 - \|\mathbf{w}_j\|^2 = 1/c$, we have

$$\sqrt{c}\, x_{j,0} = \cosh(\sqrt{c}\, d_j), \tag{36}$$

$$\sqrt{c}\,\|\mathbf{w}_j\| = \sinh(\sqrt{c}\,d_j). \tag{37}$$

Using $\mathbf{w}_j^\top \mathbf{u} = \|\mathbf{w}_j\| \cos\theta_j$, the argument of the logarithm in Eq. 2 becomes

$$\sqrt{c}\left(x_{j,0} - \mathbf{w}_j^\top \mathbf{u}\right) = \cosh(\sqrt{c}\,d_j) - \sinh(\sqrt{c}\,d_j)\cos\theta_j, \tag{38}$$

which yields Eq. 35.

(i). Since $\mathbf{w}_j \neq \mathbf{0}$, we have $d_j > 0$ and $\sinh(\sqrt{c}\,d_j) > 0$, so the argument is strictly decreasing in $\cos\theta_j$. At $\cos\theta_j = \pm 1$ it equals

$$\cosh(\sqrt{c}\,d_j) \mp \sinh(\sqrt{c}\,d_j) = e^{\mp\sqrt{c}\,d_j}. \tag{39}$$

Hence

$$e^{-\sqrt{c}\,d_j} \leq \cosh(\sqrt{c}\,d_j) - \sinh(\sqrt{c}\,d_j)\cos\theta_j \leq e^{\sqrt{c}\,d_j}, \tag{40}$$

and applying $-\frac{\alpha}{\sqrt{c}}\log(\cdot)$ gives $-\alpha d_j \leq \ell_j(\mathbf{h}) \leq \alpha d_j$, with the upper bound at $\theta_j = 0$ and the lower bound at $\theta_j = \pi$.

(ii). The logit is positive if and only if the argument is smaller than one:

$$\cosh(\sqrt{c}\,d_j) - \sinh(\sqrt{c}\,d_j)\cos\theta_j < 1. \tag{41}$$

$$\cos\theta_j > \frac{\cosh(\sqrt{c}\,d_j) - 1}{\sinh(\sqrt{c}\,d_j)}. \tag{42}$$

Using the identities

$$\cosh z - 1 = 2\sinh^2\left(\frac{z}{2}\right), \tag{43}$$

$$\sinh z = 2\sinh\left(\frac{z}{2}\right)\cosh\left(\frac{z}{2}\right), \tag{44}$$

with $z = \sqrt{c}\,d_j$, we have

$$\begin{aligned}\frac{\cosh(\sqrt{c}\,d_j) - 1}{\sinh(\sqrt{c}\,d_j)} &= \frac{2\sinh^2\left(\frac{\sqrt{c}\,d_j}{2}\right)}{2\sinh\left(\frac{\sqrt{c}\,d_j}{2}\right)\cosh\left(\frac{\sqrt{c}\,d_j}{2}\right)} \\ &= \frac{\sinh\left(\frac{\sqrt{c}\,d_j}{2}\right)}{\cosh\left(\frac{\sqrt{c}\,d_j}{2}\right)} \\ &= \tanh\left(\frac{\sqrt{c}\,d_j}{2}\right).\end{aligned} \tag{45}$$

Therefore,

$$\cos\theta_j > \tanh\left(\frac{\sqrt{c}\,d_j}{2}\right). \tag{46}$$

(iii). Setting $\cos\theta_j = 0$ gives the argument $\cosh(\sqrt{c}\,d_j) > 1$, and therefore

$$\ell_j(\mathbf{h}) = -\frac{\alpha}{\sqrt{c}}\log\cosh(\sqrt{c}\,d_j) < 0. \tag{47}$$

□

Radial specificity. Proposition 4 shows that the distance d_j of a token embedding from the pole trades off the maximum attainable logit against the range of contexts in which the token is favored. By (ii), tokens near the pole receive positive logits for nearly half of all context directions, but by (i) their logits are bounded by the small value αd_j; distant tokens can attain logits up to αd_j, but only within a narrow cone of context directions around $\mathbf{w}_j$. By (iii), every token receives a negative logit for an orthogonal context, with magnitude $\frac{\alpha}{\sqrt{c}}\log\cosh(\sqrt{c}\,d_j)$ that grows with d_j. In contrast, for a Euclidean output layer without a bias term, the logit $\mathbf{w}_j^\top \mathbf{h} = \|\mathbf{h}\|\|\mathbf{w}_j\|\cos\theta_j$ is

positive exactly when $\cos\theta_j > 0$, i.e., for half of all context directions, and is zero for orthogonal contexts, regardless of $\|\mathbf{w}_j\|$: the norm scales the logit but does not change the set of contexts in which the token is favored. LBL therefore imposes a radial, general-to-specific inductive bias over the vocabulary, consistent with the observation of He et al. (2026b) that more generic words lie closer to the pole $\boldsymbol{o}$. We present this as a structural property of LBL rather than as evidence that the learned embeddings encode a particular hierarchy.

C.2 Riemannian Gradient of LBL

We now characterize how LBL varies with the token embedding, which connects LBL to the Riemannian optimization in Appendix D. Throughout this subsection, we fix a nonzero hidden state $\mathbf{h}$ with boundary direction $\boldsymbol{\xi}_\mathbf{h} = [1, \mathbf{u}]^\top$, and write $\ell_\mathbf{h}(\boldsymbol{x}) = -\alpha b_{\boldsymbol{\xi}_\mathbf{h}}(\boldsymbol{x})$ for LBL as a function of the embedding, so that $\ell_j(\mathbf{h}) = \ell_\mathbf{h}(\boldsymbol{x}_j)$. We use the following quantities associated with a token embedding $\boldsymbol{x}_j = [x_{j,0}, \mathbf{w}_j]^\top \in \mathbb{L}_c^n$.

First, $d_j = d_\mathbb{L}^c(\boldsymbol{o}, \boldsymbol{x}_j)$ denotes the geodesic distance of $\boldsymbol{x}_j$ from the pole. Since $\sqrt{c}\, x_{j,0} = \cosh(\sqrt{c}\, d_j)$, the manifold constraint gives

$$\sinh(\sqrt{c}\, d_j) = \sqrt{c}\, \|\mathbf{w}_j\|. \tag{48}$$

Second, $\boldsymbol{v}_j$ denotes the unit tangent vector at $\boldsymbol{x}_j$ that points toward $\boldsymbol{\xi}_\mathbf{h}$. It is given by Eq. 20 with $\boldsymbol{p} = \boldsymbol{x}_j$ and $\boldsymbol{\xi} = \boldsymbol{\xi}_\mathbf{h}$,

$$\boldsymbol{v}_j = \sqrt{c}\,\big(\mu_j \boldsymbol{\xi}_\mathbf{h} - \boldsymbol{x}_j\big), \qquad \mu_j = -\frac{1}{c\langle \boldsymbol{x}_j, \boldsymbol{\xi}_\mathbf{h}\rangle_\mathcal{L}} > 0, \tag{49}$$

and satisfies $\langle \boldsymbol{v}_j, \boldsymbol{x}_j\rangle_\mathcal{L} = 0$ and $\|\boldsymbol{v}_j\|_\mathcal{L} = 1$ (Appendix B.2).

Third, σ_j denotes the geodesic through $\boldsymbol{x}_j$ with initial velocity $\boldsymbol{v}_j$,

$$\sigma_j(s) = \exp_{\boldsymbol{x}_j}^c(s\boldsymbol{v}_j) = \cosh(\sqrt{c}\, s)\, \boldsymbol{x}_j + \frac{1}{\sqrt{c}} \sinh(\sqrt{c}\, s)\, \boldsymbol{v}_j, \qquad s \in \mathbb{R}. \tag{50}$$

The parameter s is the signed geodesic distance from $\boldsymbol{x}_j$ along σ_j: $\sigma_j(s)$ is the point reached by starting at $\boldsymbol{x}_j$ and traveling a geodesic distance $|s|$ along σ_j, toward $\boldsymbol{\xi}_\mathbf{h}$ if $s > 0$ and away from $\boldsymbol{\xi}_\mathbf{h}$ if $s < 0$. In particular, $\sigma_j(0) = \boldsymbol{x}_j$, $\sigma_j'(0) = \boldsymbol{v}_j$, and $d_\mathbb{L}^c(\boldsymbol{x}_j, \sigma_j(s)) = |s|$, since σ_j has unit speed and geodesics in $\mathbb{L}_c^n$ are distance-minimizing. For $s \geq 0$, σ_j is the ray $\gamma_{\boldsymbol{x}_j, \boldsymbol{\xi}_\mathbf{h}}$ of Appendix B.2 from $\boldsymbol{x}_j$ toward $\boldsymbol{\xi}_\mathbf{h}$, so $\sigma_j(s)$ approaches $\boldsymbol{\xi}_\mathbf{h}$ as $s \to \infty$. For $\boldsymbol{x}_j = \boldsymbol{o}$, we have $\boldsymbol{v}_j = [0, \mathbf{u}]^\top$ and $\sigma_j(s) = \gamma_\mathbf{u}(s)$ for $s \geq 0$, so σ_j extends the ray $\gamma_\mathbf{u}$ in Eq. 18 from the pole to an arbitrary token embedding.

Fourth, $\mathcal{H}_j = \mathcal{H}_{\boldsymbol{\xi}_\mathbf{h}, \tau_j}$ with $\tau_j = b_{\boldsymbol{\xi}_\mathbf{h}}(\boldsymbol{x}_j)$ denotes the horosphere centered at $\boldsymbol{\xi}_\mathbf{h}$ that passes through $\boldsymbol{x}_j$, i.e., the set of embeddings that receive the same logit as $\boldsymbol{x}_j$. By property (v) in Appendix B.2, $\mathcal{H}_j = \{\boldsymbol{x} \in \mathbb{L}_c^n : \langle \boldsymbol{x}, \boldsymbol{\xi}_\mathbf{h}\rangle_\mathcal{L} = \langle \boldsymbol{x}_j, \boldsymbol{\xi}_\mathbf{h}\rangle_\mathcal{L}\}$.

Theorem 5 (Riemannian Gradient of LBL)**.** *Let $\mathbf{h}$ be a nonzero hidden state with boundary direction $\boldsymbol{\xi}_\mathbf{h}$, and let $\boldsymbol{x}_j \in \mathbb{L}_c^n$ be a token embedding. Let $\boldsymbol{v}_j \in \mathcal{T}_{\boldsymbol{x}_j}\mathbb{L}_c^n$ be the unit tangent vector at $\boldsymbol{x}_j$ that points toward $\boldsymbol{\xi}_\mathbf{h}$ (Eq. 49), i.e., the one for which the geodesic ray $s \mapsto \exp_{\boldsymbol{x}_j}^c(s\boldsymbol{v}_j)$, $s \geq 0$, converges to the boundary point represented by $\boldsymbol{\xi}_\mathbf{h}$. Let $\sigma_j(s) = \exp_{\boldsymbol{x}_j}^c(s\boldsymbol{v}_j)$, $s \in \mathbb{R}$, be the geodesic σ_j through $\boldsymbol{x}_j$ in this direction (Eq. 50), so that $\sigma_j(s)$ is the point at signed geodesic distance s from $\boldsymbol{x}_j$, toward $\boldsymbol{\xi}_\mathbf{h}$ if $s > 0$ and away from it if $s < 0$. Let $\mathcal{H}_j$ be the horosphere centered at $\boldsymbol{\xi}_\mathbf{h}$ that passes through $\boldsymbol{x}_j$. Then the following hold.*

(i) Direction toward $\boldsymbol{\xi}_\mathbf{h}$. *The Riemannian gradient $\operatorname{grad}\ell_\mathbf{h}(\boldsymbol{x}_j) \in \mathcal{T}_{\boldsymbol{x}_j}\mathbb{L}_c^n$ is $\operatorname{grad}\ell_\mathbf{h}(\boldsymbol{x}_j) = \alpha\boldsymbol{v}_j$. Since $\alpha > 0$, it points toward $\boldsymbol{\xi}_\mathbf{h}$ in the above sense, and its Lorentz norm is $\|\operatorname{grad}\ell_\mathbf{h}(\boldsymbol{x}_j)\|_\mathcal{L} = \alpha$.*

(ii) Orthogonality to the horosphere $\mathcal{H}_j$. *The Riemannian gradient $\operatorname{grad}\ell_\mathbf{h}(\boldsymbol{x}_j)$ is orthogonal to the horosphere $\mathcal{H}_j$ at $\boldsymbol{x}_j$, i.e., $\langle \operatorname{grad}\ell_\mathbf{h}(\boldsymbol{x}_j), \boldsymbol{z}\rangle_\mathcal{L} = 0$ for every tangent vector $\boldsymbol{z} \in \mathcal{T}_{\boldsymbol{x}_j}\mathcal{H}_j$.*

(iii) Update along the geodesic σ_j. *For every $\lambda \in \mathbb{R}$, the exponential map $\exp_{\boldsymbol{x}_j}^c$ satisfies $\exp_{\boldsymbol{x}_j}^c\big(\lambda \operatorname{grad}\ell_\mathbf{h}(\boldsymbol{x}_j)\big) = \sigma_j(\alpha\lambda)$, and $\ell_\mathbf{h}(\sigma_j(s)) = \ell_\mathbf{h}(\boldsymbol{x}_j) + \alpha s$ for every $s \in \mathbb{R}$. Hence a step along the Riemannian gradient $\operatorname{grad}\ell_\mathbf{h}(\boldsymbol{x}_j)$ moves $\boldsymbol{x}_j$ along the geodesic σ_j by a geodesic distance $d_\mathbb{L}^c\big(\boldsymbol{x}_j, \sigma_j(\alpha\lambda)\big) = \alpha|\lambda|$, toward $\boldsymbol{\xi}_\mathbf{h}$ if $\lambda > 0$ and away from it if $\lambda < 0$, and changes its logit by exactly $\alpha^2\lambda$.*

Proof. (i) On the open set $\{\boldsymbol{y} \in \mathbb{R}^{n+1} : \langle \boldsymbol{y}, \boldsymbol{\xi}_{\mathbf{h}} \rangle_{\mathcal{L}} < 0\}$, which contains $\mathbb{L}_c^n$ by Lemma 1, the Busemann function $b_{\boldsymbol{\xi}}(\boldsymbol{x})$ has directional derivative $\langle \boldsymbol{z}, \boldsymbol{\xi}_{\mathbf{h}} \rangle_{\mathcal{L}} / (\sqrt{c}\, \langle \boldsymbol{y}, \boldsymbol{\xi}_{\mathbf{h}} \rangle_{\mathcal{L}})$ along $\boldsymbol{z}$, so its gradient with respect to the Lorentz inner product is $\boldsymbol{\xi}_{\mathbf{h}} / (\sqrt{c}\, \langle \boldsymbol{y}, \boldsymbol{\xi}_{\mathbf{h}} \rangle_{\mathcal{L}})$. The Riemannian gradient on $\mathbb{L}_c^n$ is its orthogonal projection onto $\mathcal{T}_{\boldsymbol{x}_j} \mathbb{L}_c^n$, $\boldsymbol{z} \mapsto \boldsymbol{z} + c \langle \boldsymbol{x}_j, \boldsymbol{z} \rangle_{\mathcal{L}} \boldsymbol{x}_j$ (Nickel & Kiela, 2018), which yields

$$\begin{aligned} \operatorname{grad} b_{\boldsymbol{\xi}_{\mathbf{h}}}(\boldsymbol{x}_j) &= \frac{\boldsymbol{\xi}_{\mathbf{h}}}{\sqrt{c}\, \langle \boldsymbol{x}_j, \boldsymbol{\xi}_{\mathbf{h}} \rangle_{\mathcal{L}}} + \sqrt{c}\, \boldsymbol{x}_j \\ &= -\sqrt{c} \left(\mu_j \boldsymbol{\xi}_{\mathbf{h}} - \boldsymbol{x}_j \right) \\ &= -\boldsymbol{v}_j. \end{aligned} \tag{51}$$

Hence $\operatorname{grad} \ell_{\mathbf{h}}(\boldsymbol{x}_j) = \alpha \boldsymbol{v}_j$, and $\|\boldsymbol{v}_j\|_{\mathcal{L}} = 1$ by Appendix B.2.

(ii) Since $\mathcal{H}_j$ is cut out of $\mathbb{L}_c^n$ by $\langle \boldsymbol{x}, \boldsymbol{\xi}_{\mathbf{h}} \rangle_{\mathcal{L}} = \langle \boldsymbol{x}_j, \boldsymbol{\xi}_{\mathbf{h}} \rangle_{\mathcal{L}}$, a tangent vector $\boldsymbol{z} \in \mathcal{T}_{\boldsymbol{x}_j} \mathcal{H}_j$ satisfies $\langle \boldsymbol{z}, \boldsymbol{x}_j \rangle_{\mathcal{L}} = 0$ and $\langle \boldsymbol{z}, \boldsymbol{\xi}_{\mathbf{h}} \rangle_{\mathcal{L}} = 0$. Eq. 49 then gives $\langle \boldsymbol{v}_j, \boldsymbol{z} \rangle_{\mathcal{L}} = \sqrt{c} \left(\mu_j \langle \boldsymbol{\xi}_{\mathbf{h}}, \boldsymbol{z} \rangle_{\mathcal{L}} - \langle \boldsymbol{x}_j, \boldsymbol{z} \rangle_{\mathcal{L}} \right) = 0$, and (i) gives the claim.

(iii) By (i) and Eq. 50, $\exp_{\boldsymbol{x}_j}^c(\lambda \operatorname{grad} \ell_{\mathbf{h}}(\boldsymbol{x}_j)) = \exp_{\boldsymbol{x}_j}^c(\alpha \lambda \boldsymbol{v}_j) = \sigma_j(\alpha\lambda)$. Since $\langle \boldsymbol{\xi}_{\mathbf{h}}, \boldsymbol{\xi}_{\mathbf{h}} \rangle_{\mathcal{L}} = 0$, Eq. 49 gives $\langle \boldsymbol{v}_j, \boldsymbol{\xi}_{\mathbf{h}} \rangle_{\mathcal{L}} = -\sqrt{c}\, \langle \boldsymbol{x}_j, \boldsymbol{\xi}_{\mathbf{h}} \rangle_{\mathcal{L}}$, and using $\cosh z - \sinh z = e^{-z}$, we obtain

$$\begin{aligned} \langle \sigma_j(s), \boldsymbol{\xi}_{\mathbf{h}} \rangle_{\mathcal{L}} &= \Big(\cosh(\sqrt{c}\, s) - \sinh(\sqrt{c}\, s) \Big) \langle \boldsymbol{x}_j, \boldsymbol{\xi}_{\mathbf{h}} \rangle_{\mathcal{L}} \\ &= e^{-\sqrt{c}\, s} \langle \boldsymbol{x}_j, \boldsymbol{\xi}_{\mathbf{h}} \rangle_{\mathcal{L}}. \end{aligned} \tag{52}$$

Substituting into Busemann function $b_{\boldsymbol{\xi}}(\boldsymbol{x})$ gives $b_{\boldsymbol{\xi}_{\mathbf{h}}}(\sigma_j(s)) = b_{\boldsymbol{\xi}_{\mathbf{h}}}(\boldsymbol{x}_j) - s$, and multiplying by $-\alpha$ yields $\ell_{\mathbf{h}}(\sigma_j(s)) = \ell_{\mathbf{h}}(\boldsymbol{x}_j) + \alpha s$. Taking $s = \alpha\lambda$ gives the change $\alpha^2 \lambda$. □

In words, the Riemannian gradient $\operatorname{grad} \ell_{\mathbf{h}}(\boldsymbol{x}_j)$ points toward the boundary point $\boldsymbol{\xi}_{\mathbf{h}}$ determined by the hidden state (i), is orthogonal to the horosphere $\mathcal{H}_j$ on which $\boldsymbol{x}_j$ lies (ii), and moving $\boldsymbol{x}_j$ along it with the exponential map $\exp_{\boldsymbol{x}_j}^c$ keeps it on the geodesic σ_j, with the logit $\ell_{\mathbf{h}}$ growing by exactly α per unit geodesic distance traveled toward $\boldsymbol{\xi}_{\mathbf{h}}$ (iii). Appendix D.1 uses these properties to motivate the Riemannian optimization of the token embeddings.

D Curvature Learning Riemannian Adam (CRAdam)

This appendix presents CRAdam in detail. Appendix D.1 explains why the token embeddings of LBLM are optimized with Riemannian optimization in their spatial coordinates based on the gradient geometry of LBL in Appendix C.2. Appendix D.2 presents the Riemannian Adam update under a fixed curvature, and Appendix D.3 extends it to learnable curvature.

D.1 Motivation from the Gradient Geometry of LBL

LBLM requires each vocabulary token embedding $\boldsymbol{x}_j = [x_{j,0}, \mathbf{w}_j]^\top \in \mathbb{L}_c^n$ to remain on the Lorentz manifold during training. It stores only the spatial coordinates $\mathbf{w}_j$ and reconstructs $x_{j,0} = \sqrt{1/c + \|\mathbf{w}_j\|^2}$ when needed, so each token has the same number of learnable parameters as the vector in the Euclidean output layer.

Connection to Riemannian updates. Riemannian optimizers (Nickel & Kiela, 2018; Bécigneul & Ganea, 2019) share the same basic operation: they move a point along a geodesic with the exponential map, in a direction given by the Riemannian gradient. Theorem 5 characterizes the effect of a Riemannian gradient descent step on LBL. Consider a single position whose target token is y. For the cross entropy loss, $\partial \mathcal{L} / \partial \ell_j = p_j - \mathbf{1}[j = y]$, where p_j is the softmax probability of token j. Since $\boldsymbol{x}_j$ affects the loss only through its own logit $\ell_j(\mathbf{h})$, Theorem 5 (i) gives the Riemannian gradient of the loss with respect to $\boldsymbol{x}_j$ as $(p_j - \mathbf{1}[j = y])\, \alpha \boldsymbol{v}_j$. A Riemannian gradient descent step therefore moves the target embedding along σ_y toward $\boldsymbol{\xi}_{\mathbf{h}}$, which raises its logit, and moves every other embedding along its own σ_j away from $\boldsymbol{\xi}_{\mathbf{h}}$, which lowers its logit.

Because the exponential map moves each embedding exactly along σ_j, the size of this change can be computed exactly. By Theorem 5 (iii), a step with learning rate η changes the logit by

$$\Delta \ell_j(\mathbf{h}) = -\eta \alpha^2 \big(p_j - \mathbf{1}[j = y] \big) \tag{53}$$

for any value of η; for example, doubling the learning rate doubles the change. If the embedding were instead moved along a straight line in its spatial coordinates and then placed back on the manifold, it would in general leave geodesic σ_j, and Eq. 53 would hold only approximately, with an error that grows as the step becomes larger.

These statements hold exactly only for one context and one step. In training, the gradient of a token sums contributions from many contexts with different boundary directions $\boldsymbol{\xi}_{\mathbf{h}}$, and Riemannian Adam also accumulates gradients from earlier steps in its first moment, so the actual update follows these geodesics only approximately. We therefore use this analysis to motivate updating token embeddings along geodesics with Riemannian optimization, not to describe the full training dynamics.

Shared spatial parameterization. LBL and CRAdam share the same parameterization of the token embeddings. LBL is computed from the spatial coordinates through $x_{j,0} - \mathbf{w}_j^\top \mathbf{u}$ with $x_{j,0} = \sqrt{1/c + \|\mathbf{w}_j\|^2}$, and CRAdam stores and updates only $\mathbf{w}_j$. In these coordinates, the Lorentz metric at $\boldsymbol{x}_j$ and its inverse are

$$G_j = I - \frac{c\,\mathbf{w}_j\mathbf{w}_j^\top}{1 + c\|\mathbf{w}_j\|^2}, \tag{54}$$

$$G_j^{-1} = I + c\,\mathbf{w}_j\mathbf{w}_j^\top, \tag{55}$$

which are Eqs. 60 and 61 evaluated at $\mathbf{w}_j$. The Euclidean gradient of LBL is

$$\nabla_{\mathbf{w}_j}\ell_{\mathbf{h}} = \frac{\alpha}{\sqrt{c}}\,\frac{\mathbf{u} - \mathbf{w}_j/x_{j,0}}{x_{j,0} - \mathbf{w}_j^\top \mathbf{u}}, \tag{56}$$

and the spatial component of the Riemannian gradient in Theorem 5(i) is

$$\big(\alpha \boldsymbol{v}_j\big)_{\mathrm{s}} = \frac{\alpha}{\sqrt{c}}\,\frac{\mathbf{u}}{x_{j,0} - \mathbf{w}_j^\top \mathbf{u}} - \alpha\sqrt{c}\,\mathbf{w}_j = G_j^{-1}\nabla_{\mathbf{w}_j}\ell_{\mathbf{h}}. \tag{57}$$

For $\mathbf{w}_j \neq \mathbf{0}$, let $P_j = \mathbf{w}_j\mathbf{w}_j^\top/\|\mathbf{w}_j\|^2$ denote the projection onto the radial direction. Then $G_j^{-1} = (I - P_j) + (1 + c\|\mathbf{w}_j\|^2)P_j$, and by Eq. 48, $1 + c\|\mathbf{w}_j\|^2 = \cosh^2(\sqrt{c}\,d_j)$. Relative to the Riemannian gradient, the Euclidean gradient therefore scales down its radial component by $\cosh^2(\sqrt{c}\,d_j)$, a factor that grows exponentially with the distance of the token from the pole, while leaving the orthogonal component unchanged. For example, for a context aligned with the token, $\mathbf{u} = \mathbf{w}_j/\|\mathbf{w}_j\|$, the Euclidean gradient is purely radial with norm $\alpha/\cosh(\sqrt{c}\,d_j)$, whereas the Riemannian gradient has metric norm α. Moreover, in these coordinates, a radial displacement of fixed Euclidean length corresponds to a geodesic displacement that shrinks by the factor $\cosh(\sqrt{c}\,d_j)$.

An optimizer that treats $\mathbf{w}_j$ as Euclidean parameters thus both receives distorted gradients and measures its steps in a way that depends on the position of the token rather than on the geometry of LBL. This distortion acts along the direction of $\mathbf{w}_j$, which is generally not a coordinate axis, and therefore cannot be corrected by the coordinate-wise scaling of Adam. CRAdam removes it by applying G_j^{-1} to the gradient (Eq. 62) and by measuring its steps with the metric norm (Eq. 68). Consistent with this, Table 6 shows that optimizing the spatial coordinates of LBLM with Adam yields 36.19 with fixed curvature and 37.69 with learnable curvature, both below the Euclidean baseline (38.75, Table 1), whereas CRAdam with fixed curvature reaches 40.75, above it. These observations motivate optimizing the token embeddings with Riemannian Adam in the spatial parameterization (Appendix D.2), extended to learnable curvature (Appendix D.3).

D.2 Fixed-Curvature Riemannian Adam

For notational simplicity, we omit the token index j in the following and write $\boldsymbol{x} = [x_0, \mathbf{w}]^\top$ for an arbitrary vocabulary token embedding. Let $\mathbf{W} \in \mathbb{R}^{|\mathcal{V}|\times n}$ collect the spatial coordinates of all vocabulary token embeddings. Using LBL in Eq. 2, the language modeling objective is

$$\mathcal{L} = \mathcal{L}_{\mathrm{CE}}(\mathbf{W}, c). \tag{58}$$

At optimization step t, the Euclidean gradient with respect to the spatial coordinates of an arbitrary token is denoted by

$$\mathbf{g}_t = \nabla_{\mathbf{w}_t}\mathcal{L}. \tag{59}$$

We first consider the update of the vocabulary representations while holding the curvature fixed at $-c_t$.

For the current spatial coordinate $\mathbf{w}_t$, the induced Lorentz metric is

$$G_t = I - \frac{c_t \mathbf{w}_t \mathbf{w}_t^\top}{1 + c_t \|\mathbf{w}_t\|^2}, \tag{60}$$

whose inverse is

$$G_t^{-1} = I + c_t \mathbf{w}_t \mathbf{w}_t^\top. \tag{61}$$

The corresponding Riemannian gradient is therefore

$$\mathbf{r}_t = G_t^{-1} \mathbf{g}_t = \mathbf{g}_t + c_t \left(\mathbf{w}_t^\top \mathbf{g}_t\right) \mathbf{w}_t. \tag{62}$$

For each token, we maintain the spatial coordinate $\mathbf{m}_t \in \mathbb{R}^n$ of the first moment and a second moment $v_t \in \mathbb{R}$. The first moment $\mathbf{m}_t$ is the spatial component of the tangent vector $\boldsymbol{\mu}_t \in \mathcal{T}_{\boldsymbol{x}_t} \mathbb{L}^n_{c_t}$ at the current point $\boldsymbol{x}_t \in \mathbb{L}^n_{c_t}$, such that $\mathbf{m}_t = (\boldsymbol{\mu}_t)_{\mathrm{s}}$. Since the current Riemannian gradient is defined in tangent space $\mathcal{T}_{\boldsymbol{x}_t} \mathbb{L}^n_{c_t}$, the first moment from the preceding optimization step has already been parallel transported to the tangent space at $\boldsymbol{x}_t$ at the end of that step before the current moment update.

The Adam moments are updated as

$$\mathbf{m}_t^+ = \beta_1 \mathbf{m}_t + (1 - \beta_1) \mathbf{r}_t, \tag{63}$$

$$v_t^+ = \beta_2 v_t + (1 - \beta_2) \|\mathbf{r}_t\|^2_{G_t}, \tag{64}$$

where the intrinsic squared norm of the Riemannian gradient can be written as

$$\|\mathbf{r}_t\|^2_{G_t} = \mathbf{r}_t^\top G_t \mathbf{r}_t = \|\mathbf{g}_t\|^2 + c_t \left(\mathbf{w}_t^\top \mathbf{g}_t\right)^2. \tag{65}$$

Thus, following standard Riemannian Adam, the second moment is maintained for each token to provide adaptive scaling for the first moment (Kingma & Ba, 2014).

After bias correction,

$$\widehat{\mathbf{m}}_t = \frac{\mathbf{m}_t^+}{1 - \beta_1^{t+1}}, \qquad \widehat{v}_t = \frac{v_t^+}{1 - \beta_2^{t+1}}, \tag{66}$$

the adaptive Riemannian direction is

$$\mathbf{d}_t = \frac{\widehat{\mathbf{m}}_t}{\sqrt{\widehat{v}_t} + \epsilon}. \tag{67}$$

Its squared norm under the spatial Lorentz metric is

$$\|\mathbf{d}_t\|^2_{G_t} = \|\mathbf{d}_t\|^2 - \frac{c_t \left(\mathbf{w}_t^\top \mathbf{d}_t\right)^2}{1 + c_t \|\mathbf{w}_t\|^2}. \tag{68}$$

Given a learning rate η, we compute $\rho_t = \eta \sqrt{c_t}\, \|\mathbf{d}_t\|_{G_t}$ to ensure stable manifold updates, along with the clipping factor: $\psi_t = \min\left(1, \frac{\rho_{\max}}{\rho_t + \epsilon_{\mathrm{clip}}}\right)$. The resulting tangent step is then given by $\boldsymbol{\delta}_t = -\eta \psi_t \mathbf{d}_t$.

The token embedding is then updated on the fixed curvature Lorentz manifold using the exponential map in Eq. 9. Let $\bar{\boldsymbol{\delta}}_t = \left[\mathbf{w}_t^\top \boldsymbol{\delta}_t / x_{0,t},\ \boldsymbol{\delta}_t\right]^\top \in \mathcal{T}_{\boldsymbol{x}_t} \mathbb{L}^n_{c_t}$ denote the tangent vector whose spatial component is $\boldsymbol{\delta}_t$. Its Lorentz norm equals the norm of $\boldsymbol{\delta}_t$ under the induced spatial metric, $\|\bar{\boldsymbol{\delta}}_t\|_{\mathcal{L}} = \|\boldsymbol{\delta}_t\|_{G_t}$.

We therefore define

$$z_t = \sqrt{c_t}\, \|\boldsymbol{\delta}_t\|_{G_t}, \tag{69}$$

which is precisely the argument of the hyperbolic functions in Eqs. 9 and 10. Taking the spatial component of the exponential map gives

$$\widetilde{\mathbf{w}}_{t+1} = \cosh(z_t) \mathbf{w}_t + \mathrm{sinhc}(z_t) \boldsymbol{\delta}_t, \tag{70}$$

where $\mathrm{sinhc}(z) = \sinh(z)/z$ and $\mathrm{sinhc}(0) = 1$.

Throughout this section, quantities marked by $\widetilde{\cdot}$ denote intermediate states obtained after the fixed curvature Riemannian update under c_t but before the transition from c_t to c_{t+1}.

Since $\mathbf{m}_t^+$ is associated with the tangent space at $\boldsymbol{x}_t$, the movement of the token embedding from $\boldsymbol{x}_t$ to $\widetilde{\boldsymbol{x}}_{t+1}$ requires the first moment to be parallel transported to the tangent space at the updated point. The full tangent vector $\boldsymbol{\mu}_t^+$ corresponding to $\mathbf{m}_t^+$ is reconstructed as

$$\boldsymbol{\mu}_t^+ = \begin{bmatrix} \dfrac{\mathbf{w}_t^\top \mathbf{m}_t^+}{x_{0,t}} \\ \mathbf{m}_t^+ \end{bmatrix}, \qquad x_{0,t} = \sqrt{\frac{1}{c_t} + \|\mathbf{w}_t\|^2}. \tag{71}$$

Parallel transport along the geodesic from $\boldsymbol{x}_t$ to $\widetilde{\boldsymbol{x}}_{t+1}$ gives

$$\widetilde{\boldsymbol{\mu}}_{t+1} = \boldsymbol{\mu}_t^+ + \frac{\langle \widetilde{\boldsymbol{x}}_{t+1}, \boldsymbol{\mu}_t^+ \rangle_{\mathcal{L}}}{c_t^{-1} - \langle \boldsymbol{x}_t, \widetilde{\boldsymbol{x}}_{t+1} \rangle_{\mathcal{L}}} \left(\boldsymbol{x}_t + \widetilde{\boldsymbol{x}}_{t+1} \right). \tag{72}$$

We retain its spatial component as $\widetilde{\mathbf{m}}_{t+1} = \left(\widetilde{\boldsymbol{\mu}}_{t+1}\right)_{\mathrm{s}}$. This operation can also be evaluated directly in spatial coordinates. Let

$$A_t = \widetilde{\mathbf{w}}_{t+1}^\top \mathbf{m}_t^+ - \frac{\widetilde{x}_{0,t+1}}{x_{0,t}} \mathbf{w}_t^\top \mathbf{m}_t^+, \tag{73}$$

$$D_t = c_t^{-1} + x_{0,t} \widetilde{x}_{0,t+1} - \mathbf{w}_t^\top \widetilde{\mathbf{w}}_{t+1}. \tag{74}$$

Then

$$\widetilde{\mathbf{m}}_{t+1} = \mathrm{PT}^{c_t}_{\mathbf{w}_t \to \widetilde{\mathbf{w}}_{t+1}}(\mathbf{m}_t^+) = \mathbf{m}_t^+ + \frac{A_t}{D_t} \left(\mathbf{w}_t + \widetilde{\mathbf{w}}_{t+1} \right). \tag{75}$$

Thus, the full tangent vector need not be stored.

For the second moment, we have:

$$\widetilde{v}_{t+1} = v_t^+. \tag{76}$$

This completes the update under the current curvature. The resulting $\widetilde{\mathbf{w}}_{t+1}$, $\widetilde{\mathbf{m}}_{t+1}$, and $\widetilde{v}_{t+1}$ therefore constitute the intermediate parameter and optimizer states after a fixed curvature Riemannian Adam update under c_t.

D.3 Riemannian Adam with Learnable Curvature

Standard Riemannian Adam assumes that consecutive parameter updates are performed under a fixed manifold geometry. In this study, the curvature is updated as a learnable parameter. Following Desai et al. (2023), we parameterize the curvature in the log domain as $\omega_t = \log c_t$, so that $c_t = \exp(\omega_t) > 0$. The log curvature parameter ω_t is optimized using SGD without weight decay. During the curvature update, we update the unconstrained parameter ω_t using the partial derivative

$$\frac{\partial \mathcal{L}_t}{\partial \omega_t} = c_t \frac{\partial \mathcal{L}_t}{\partial c_t}, \tag{77}$$

which is taken with the spatial coordinates $\mathbf{W}$ held fixed, since the time coordinates are recomputed from $\mathbf{W}$ and c_t in the forward pass. We then recover the updated curvature as $c_{t+1} = \exp(\omega_{t+1})$.

We denote the resulting curvature increment by

$$\Delta c_t = c_{t+1} - c_t = c_t \operatorname{expm1}(\omega_{t+1} - \omega_t), \tag{78}$$

where $\operatorname{expm1}(x) = \exp(x) - 1$. Evaluating Δc_t from the log-curvature increment, rather than as the difference between $\exp(\omega_{t+1})$ and $\exp(\omega_t)$, avoids cancellation when a single update changes the curvature only slightly.

D.3.1 Point Transition under Learnable Curvature

After the fixed curvature Riemannian update, the intermediate point $\widetilde{\boldsymbol{x}}_{t+1}$ is defined on $\mathbb{L}^n_{c_t}$ under curvature c_t, whereas the next optimization step operates on $\mathbb{L}^n_{c_{t+1}}$. Since LBLM stores only the spatial coordinates of each Lorentz point, we maintain the intermediate spatial coordinate across the curvature update,

$$\mathbf{w}_{t+1} = \widetilde{\mathbf{w}}_{t+1}, \tag{79}$$

and recompute the corresponding time coordinate according to the Lorentz constraint under the updated curvature:

$$x_{0,t+1} = \sqrt{\frac{1}{c_{t+1}} + \|\mathbf{w}_{t+1}\|^2}. \tag{80}$$

Accordingly,

$$\boldsymbol{x}_{t+1} = \begin{bmatrix} x_{0,t+1} \\ \mathbf{w}_{t+1} \end{bmatrix} \in \mathbb{L}^n_{c_{t+1}}, \tag{81}$$

which satisfies

$$\langle \boldsymbol{x}_{t+1}, \boldsymbol{x}_{t+1} \rangle_{\mathcal{L}} = -\frac{1}{c_{t+1}}. \tag{82}$$

Thus, the spatial coordinate is preserved across the curvature transition, $\mathbf{w}_{t+1} = \widetilde{\mathbf{w}}_{t+1}$, while the full Lorentz point is reconstructed to satisfy the constraint associated with the updated curvature c_{t+1}. This construction carries the point from $\mathbb{L}^n_{c_t}$ to $\mathbb{L}^n_{c_{t+1}}$ through the spatial coordinate $\mathbf{w}_{t+1}$. Prior studies (Chami et al., 2019; Bdeir et al., 2026) instead map the point to a tangent space with the logarithmic map under the previous curvature c_t, and then back to the Lorentz manifold with the exponential map under the updated curvature c_{t+1}. Our transition avoids both the logarithmic and exponential maps and only recomputes the time coordinate from the preserved spatial coordinate to satisfy the Lorentz manifold constraint. Since this transition is performed within every optimization step immediately after the curvature update, the vocabulary parameters of the LLM output layer remain valid Lorentz points after each curvature update, and every token embedding used in the next forward pass lies on $\mathbb{L}^n_{c_{t+1}}$.

Consistency with the curvature gradient. The spatial coordinates are preserved because of how LBLM is parameterized. The learnable parameters of the output layer are the spatial coordinates $\mathbf{W}$ and the log-curvature ω, and the time coordinate of each token embedding is recomputed from $\mathbf{w}$ and c in the forward pass. Consequently, the curvature gradient $g_{\omega,t} = \partial \mathcal{L}_t / \partial \omega_t$ is a partial derivative taken with $\mathbf{W}$ held fixed: it measures how the loss changes when the curvature varies while the spatial coordinates remain unchanged. To first order, a joint update of $(\mathbf{W}, \omega)$ changes the loss by

$$\mathcal{L}(\mathbf{W}_{t+1}, \omega_{t+1}) - \mathcal{L}(\mathbf{W}_t, \omega_t) \approx \left\langle \nabla_{\mathbf{W}} \mathcal{L}_t, \mathbf{W}_{t+1} - \mathbf{W}_t \right\rangle + g_{\omega,t} \, (\omega_{t+1} - \omega_t), \tag{83}$$

where the first term is controlled by the Riemannian update of the embeddings and the second by the gradient step on ω. If the transition additionally modified the spatial coordinates, it would introduce a further change of $\mathbf{W}$ that is controlled by neither step, and the effect of the curvature update would no longer be the one measured by $g_{\omega,t}$. This yields the following principle: the quantity preserved during a curvature transition should be the one held fixed when the curvature gradient is computed. Since LBLM is parameterized by spatial coordinates, CRAdam preserves $\mathbf{w}$, so that the embedding update under c_t and the curvature update together form a joint step on $(\mathbf{W}, \omega)$ whose first-order effect is given by Eq. 83.

Difference from transfer through the tangent space at the origin. For a token embedding $\boldsymbol{x}_j \in \mathbb{L}^n_c$, let $d_j = d^c_{\mathbb{L}}(\boldsymbol{o}, \boldsymbol{x}_j)$ denote its geodesic distance from the pole $\boldsymbol{o} = [1/\sqrt{c}, \mathbf{0}]^\top$ under the current curvature c. Since $\sqrt{c}\, x_{j,0} = \cosh(\sqrt{c}\, d_j)$, the manifold constraint gives

$$\sinh(\sqrt{c}\, d_j) = \sqrt{c}\, \|\mathbf{w}_j\|, \tag{84}$$

so d_j depends on c even when $\mathbf{w}_j$ is fixed. Bdeir et al. (2026) transfer parameters between curvatures through the tangent space at the origin, applying $\log_{\boldsymbol{o}}^{c_t}$ followed by $\exp_{\boldsymbol{o}}^{c_{t+1}}$; Chami et al. (2019) use the same route for features between layers. Since $\mathcal{T}_{\boldsymbol{o}} \mathbb{L}^n_c = \{\boldsymbol{v} : v_0 = 0\}$ does not depend on c, this transfer preserves the tangent vector $\log_{\boldsymbol{o}}^{c_t}(\boldsymbol{x})$, i.e., the direction of the embedding and its distance from the pole, measured on $\mathbb{L}^n_{c_t}$ before and on $\mathbb{L}^n_{c_{t+1}}$ after the transition, whereas CRAdam preserves its spatial coordinates. By Eq. 84, the two differ whenever the curvature changes. CRAdam nevertheless preserves the angles between token embeddings at the pole and the ordering of their distances from the pole, since the direction of $\mathbf{w}_j$ is unchanged and, for a fixed curvature, d_j increases monotonically with $\|\mathbf{w}_j\|$; it does not preserve the distances themselves. Under our parameterization, preserving the distance would therefore change $\mathbf{w}_j$ during the transition and depart

from the principle above. As an example, consider a context whose direction is aligned with a token, $\mathbf{u} = \mathbf{w}_j/\|\mathbf{w}_j\|$ with $\mathbf{w}_j \neq \mathbf{0}$. Then $\boldsymbol{x}_j = \gamma_{\mathbf{u}}(d_j)$, and Proposition 3 gives $\ell_j(\mathbf{h}) = \alpha d_j$, so that

$$\left.\frac{\partial \ell_j(\mathbf{h})}{\partial \omega}\right|_{\mathbf{w}_j} = \frac{\alpha}{2\sqrt{c}}\Big(\tanh(\sqrt{c}\, d_j) - \sqrt{c}\, d_j\Big) < 0, \tag{85}$$

where d_j is evaluated at the current curvature. This derivative enters $g_{\omega,t}$ through the chain rule, whereas under a transfer that preserves d_j, the logit $\ell_j(\mathbf{h}) = \alpha d_j$ does not change with the curvature at all. The change of this logit anticipated by the curvature gradient is thus realized by CRAdam but not by the origin-based transfer. This does not make the origin-based transfer inconsistent in general: for a model parameterized by tangent vectors at the origin, the curvature gradient is taken with these vectors fixed, and preserving them is the consistent choice. The appropriate transition is determined by the parameterization, and for the spatial parameterization of LBLM it is the one adopted by CRAdam.

Relation to recomputing the time coordinate. A curvature transition should keep the embeddings valid on the updated manifold and keep the optimizer state consistent with the updated geometry; otherwise, invalid intermediate states or mismatched momenta can degrade training (Bdeir et al., 2026). The point transition of CRAdam meets these requirements through three properties. First, it is performed within every optimization step immediately after the curvature update, so every embedding used in the next forward pass lies on $\mathbb{L}^n_{c_{t+1}}$ and no invalid intermediate state arises. Second, the accumulated optimizer state is adjusted explicitly: since the point keeps its spatial coordinates, only the metric coefficient at the same $\mathbf{w}_{t+1}$ changes with the curvature, so the first moment is corrected by the closed form radial rescaling in Appendix D.3.3. Third, preserving the spatial coordinates is not a projection applied after the fact but follows from the consistency with the curvature gradient discussed above. Computationally, it only recomputes the time coordinate by Eq. 80, without logarithmic or exponential maps.

D.3.2 Moments Transition under Learnable Curvature

The point transition above ensures that each token embedding satisfies the Lorentz manifold constraint after the curvature is updated from c_t to c_{t+1}. However, Riemannian Adam also maintains optimizer states whose geometric interpretation depends on the Riemannian metric. Since the spatial metric depends on the curvature, a curvature update changes the metric governing these optimizer states. When $c_t \neq c_{t+1}$, the metric at $\mathbf{w}_{t+1}$ therefore changes as

$$G_{c_t}(\mathbf{w}_{t+1}) \neq G_{c_{t+1}}(\mathbf{w}_{t+1}). \tag{86}$$

The intermediate first moment $\widetilde{\mathbf{m}}_{t+1}$ is the spatial component of the transported tangent vector $\widetilde{\boldsymbol{\mu}}_{t+1} \in \mathcal{T}_{\widetilde{\boldsymbol{x}}_{t+1}}\mathbb{L}^n_{c_t}$, where $\widetilde{\boldsymbol{x}}_{t+1} \in \mathbb{L}^n_{c_t}$ is the intermediate point obtained under curvature c_t with spatial coordinate $\widetilde{\mathbf{w}}_{t+1} = \mathbf{w}_{t+1}$. Accordingly, its spatial metric before the curvature transition is $G_{c_t}(\mathbf{w}_{t+1})$. It must therefore be adjusted before being used under $G_{c_{t+1}}(\mathbf{w}_{t+1})$. In contrast, the second moment accumulates the squared metric norms of historical Riemannian gradients. It therefore does not require the corresponding directional transformation. We retain it unchanged as

$$v_{t+1} = \widetilde{v}_{t+1}. \tag{87}$$

For the transition of $\widetilde{\mathbf{m}}_{t+1}$, let

$$q_{t+1} = \|\mathbf{w}_{t+1}\|^2, \qquad P_{t+1} = \frac{\mathbf{w}_{t+1}\mathbf{w}_{t+1}^\top}{q_{t+1}}, \tag{88}$$

where P_{t+1} is the Euclidean projection onto the radial direction defined by $\mathbf{w}_{t+1}$. Using Eq. 60, the spatial Lorentz metric can be decomposed as

$$G_c(\mathbf{w}_{t+1}) = (I - P_{t+1}) + \frac{1}{1 + cq_{t+1}} P_{t+1}. \tag{89}$$

For any spatial vector $\mathbf{a} \in \mathbb{R}^n$, let $\mathbf{a}^{(r)} = P_{t+1}\mathbf{a}$ and $\mathbf{a}^{(o)} = (I - P_{t+1})\mathbf{a}$ denote its radial and orthogonal components with respect to $\mathbf{w}_{t+1}$, respectively. Then Eq. 89 gives

$$G_c(\mathbf{w}_{t+1})\mathbf{a} = \mathbf{a}^{(o)} + \frac{1}{1 + cq_{t+1}}\mathbf{a}^{(r)}. \tag{90}$$

Hence, the orthogonal component $\mathbf{a}^{(o)}$ is unaffected by the curvature, whereas the radial component $\mathbf{a}^{(r)}$ is scaled by $1/(1 + cq_{t+1})$. Therefore, changing the curvature modifies only the metric along the radial direction defined by $\mathbf{w}_{t+1}$.

Accordingly, we decompose the intermediate first moment $\widetilde{\mathbf{m}}_{t+1}$ as

$$\widetilde{\mathbf{m}}_{t+1} = \widetilde{\mathbf{m}}^{(o)}_{t+1} + \widetilde{\mathbf{m}}^{(r)}_{t+1}, \tag{91}$$

where

$$\widetilde{\mathbf{m}}^{(o)}_{t+1} = (I - P_{t+1})\,\widetilde{\mathbf{m}}_{t+1}, \qquad \widetilde{\mathbf{m}}^{(r)}_{t+1} = P_{t+1}\widetilde{\mathbf{m}}_{t+1} = \frac{\mathbf{w}_{t+1}^{\top}\widetilde{\mathbf{m}}_{t+1}}{q_{t+1}}\mathbf{w}_{t+1}. \tag{92}$$

The orthogonal component $\widetilde{\mathbf{m}}^{(o)}_{t+1}$ retains the same metric length during the curvature update and therefore requires no adjustment. For the radial component $\widetilde{\mathbf{m}}^{(r)}_{t+1}$, its squared metric norm under c_t is

$$\left\|\widetilde{\mathbf{m}}^{(r)}_{t+1}\right\|^2_{G_{c_t}} = \frac{\left\|\widetilde{\mathbf{m}}^{(r)}_{t+1}\right\|^2}{1 + c_t q_{t+1}}. \tag{93}$$

We seek a transformed first moment of the form

$$\mathbf{m}_{t+1} = \widetilde{\mathbf{m}}^{(o)}_{t+1} + \kappa_t \widetilde{\mathbf{m}}^{(r)}_{t+1}, \tag{94}$$

where only the radial component is rescaled.

D.3.3 First Moment Transition

Changing the curvature can make optimizer states defined under the previous geometry inconsistent with the updated manifold. Bdeir et al. (2026) transfer hyperbolic parameters and their associated momentum states between hyperbolic space with different curvatures through the tangent space at the origin. We instead adopt a transition tailored to the spatial coordinates of the token embeddings.

A straightforward alternative is to leave the first moment unchanged during the curvature transition, i.e., $\mathbf{m}_{t+1} = \widetilde{\mathbf{m}}_{t+1}$. However, even though no new gradient enters during the transition, the metric norm of the first moment would still change. By Eqs. 90 and 93,

$$\|\widetilde{\mathbf{m}}_{t+1}\|^2_{G_{c_{t+1}}} = \left\|\widetilde{\mathbf{m}}^{(o)}_{t+1}\right\|^2 + \frac{1 + c_t q_{t+1}}{1 + c_{t+1} q_{t+1}} \left\|\widetilde{\mathbf{m}}^{(r)}_{t+1}\right\|^2_{G_{c_t}}, \tag{95}$$

so the metric norm of the radial component $\widetilde{\mathbf{m}}^{(r)}_{t+1}$ would decrease when the curvature increases and increase when it decreases. This change would carry over to the next update. The adaptive direction normalizes the first moment by $\sqrt{\widehat{v}}$ (Eq. 67). Hence, the curvature update alone would rescale the radial part of the subsequent step, which governs how far each token embedding moves toward or away from the origin. This variant corresponds to CRAdam w/o MT in Table 6, which reduces the average score from 41.78 to 40.40.

To avoid this effect, we follow the principle of parallel transport, which preserves the metric norm of the first moment when moving it to the updated point at fixed curvature (Bécigneul & Ganea, 2019). We maintain this principle when the curvature changes and require the first moment to retain its metric norm, so that its magnitude relative to the retained second moment is unchanged. Since only the radial metric coefficient depends on the curvature, we retain the orthogonal component unchanged and rescale the radial component by a factor κ_t, chosen to preserve the metric norm of the first moment.

Specifically, we impose

$$\|\mathbf{m}_{t+1}\|_{G_{c_{t+1}}} = \|\widetilde{\mathbf{m}}_{t+1}\|_{G_{c_t}}. \tag{96}$$

As shown by the radial–orthogonal decomposition above, only the metric coefficient associated with the radial component $\widetilde{\mathbf{m}}^{(r)}_{t+1}$ changes with the curvature. The metric-norm-preserving transition can be achieved by rescaling only the radial component $\widetilde{\mathbf{m}}^{(r)}_{t+1}$. Accordingly, substituting Eq. 94 into Eq. 96 gives

$$\frac{\kappa_t^2 \left\|\widetilde{\mathbf{m}}^{(r)}_{t+1}\right\|^2}{1 + c_{t+1} q_{t+1}} = \frac{\left\|\widetilde{\mathbf{m}}^{(r)}_{t+1}\right\|^2}{1 + c_t q_{t+1}}. \tag{97}$$

Table 8: **Training hyperparameters and model depths.** LR denotes learning rate.

Depth	c_0	c lr	Output Layer LR	c Update Start	Initial Spatial Norm	Initial α	α LR
12	0.05	1×10^{-4}	0.30	50%	$0.90/\sqrt{c_0}$	5.0	1×10^{-2}
16	0.05	1×10^{-4}	0.30	60%	$0.95/\sqrt{c_0}$	5.0	1×10^{-2}
20	0.05	1×10^{-4}	0.20	50%	$0.90/\sqrt{c_0}$	5.0	1×10^{-2}
24	0.05	1×10^{-4}	0.20	50%	$0.95/\sqrt{c_0}$	5.0	1×10^{-2}

A positive radial scaling factor satisfying this condition is

$$\kappa_t = \sqrt{\frac{1 + c_{t+1} q_{t+1}}{1 + c_t q_{t+1}}}. \tag{98}$$

Note that κ_t^2 is exactly the reciprocal of the radial factor in Eq. 95; the rescaling thus precisely compensates for the norm change that would occur if the first moment were left unchanged. Hence, the first moment transition is

$$\mathbf{m}_{t+1} = \widetilde{\mathbf{m}}_{t+1}^{(o)} + \kappa_t \widetilde{\mathbf{m}}_{t+1}^{(r)}. \tag{99}$$

Since $\kappa_t - 1 = (\kappa_t^2 - 1)/(\kappa_t + 1)$ and, by Eq. 98,

$$\kappa_t^2 - 1 = \frac{\Delta c_t \, q_{t+1}}{1 + c_t q_{t+1}}, \tag{100}$$

the transition in Eq. 99 can be written equivalently in spatial coordinates as

$$\mathbf{m}_{t+1} = \widetilde{\mathbf{m}}_{t+1} + \chi_t \left(\mathbf{w}_{t+1}^\top \widetilde{\mathbf{m}}_{t+1} \right) \mathbf{w}_{t+1}, \qquad \chi_t = \frac{\Delta c_t}{(1 + c_t q_{t+1})(1 + \kappa_t)}. \tag{101}$$

This expression involves no division by q_{t+1}, its denominator is always greater than one, and it requires no stabilizing constant. It also avoids the cancellation that arises from evaluating $\kappa_t - 1$ directly when $c_{t+1} \approx c_t$. Although the radial–orthogonal decomposition assumes $q_{t+1} > 0$, Eq. 101 remains well defined at $q_{t+1} = 0$, where the correction term vanishes; this is consistent with the metric $G_c(\mathbf{0}) = I$ being independent of the curvature. When $c_{t+1} = c_t$, we have $\Delta c_t = 0$ and hence $\chi_t = 0$, so both the point and moment transitions reduce exactly to the identity, recovering the fixed curvature Riemannian Adam update.

The complete update therefore first performs Riemannian Adam on $\mathbb{L}_{c_t}^n$ to obtain the intermediate states $\widetilde{\mathbf{w}}_{t+1}$, $\widetilde{\mathbf{m}}_{t+1}$, and $\widetilde{v}_{t+1}$, and then makes these states consistent with the updated curvature c_{t+1}. During the curvature transition, the spatial coordinate of the token embedding is preserved as $\mathbf{w}_{t+1} = \widetilde{\mathbf{w}}_{t+1}$, while the first moment preserves its orthogonal component and rescales only its radial component to maintain its metric norm under the updated spatial metric. The second moment is retained as the accumulated statistic of squared Riemannian gradient norms. This yields a closed-form curvature transition in spatial coordinates without additional cross-curvature logarithmic map or exponential map operations.

E Training Configurations

E.1 Experimental Settings

Both the Euclidean baseline and LBLM are implemented within the NanoChat codebase (Karpathy, 2025). Results reported over three runs are obtained from independent runs with random seeds $\{41, 42, 43\}$ and are presented as mean $\pm$ standard deviation. Unless otherwise specified, all ablation studies are conducted on the depth-12 model and are obtained from a single run with random seed 43.

Following the official NanoChat configuration used in our experiments, we use a maximum sequence length of 2048, an aspect ratio of 64, and a head dimension of 128 across all model depths. The

training budget is set to 12 times the number of scaling parameters in the Transformer blocks and output layer. The input and output embeddings are not tied. The input embedding layer is optimized with AdamW using a learning rate of 0.3, $\beta_1 = 0.8$, $\beta_2 = 0.995$, $\epsilon = 10^{-10}$, and a weight decay of 0.001. For the output layer, the Euclidean baseline is optimized with Adam and the hyperbolic models with RAdam or the corresponding RAdam variants (e.g., CRAdam and Bdeir-RAdam), all using $\beta_1 = 0.8$, $\beta_2 = 0.96$, and $\epsilon = 10^{-10}$. The matrix learning rate and scalar learning rate are set to 0.02 and 0.5, respectively, with a weight decay of 0.28. We use 40 warmup steps and decay the learning rate over the final 65% of training, reaching 0.05 times its configured value at the end of training.

Appendix E reports the hyperparameters for each backbone and model depth. The Euclidean baseline and LBLM share all training hyperparameters that apply to both models. LBLM additionally introduces hyperparameters specific to its hyperbolic output layer. Here, c_0 denotes the initial curvature, and c lr denotes the learning rate used for curvature optimization. The curvature update start specifies when curvature updates begin, expressed as a percentage of the total training steps. For example, 60% means that the curvature remains fixed at c_0 during the first 60% of training steps and is updated during the remaining 40%. The initial spatial norm specifies the norm of the spatial coordinates of each token embedding at initialization. For example, $0.90/\sqrt{c_0}$ means that this norm is initialized to 90% of the initial curvature radius $1/\sqrt{c_0}$. The table also reports the initial value and learning rate of the logit scale parameter α.

HELM training hyperparameters. To provide a comparison with HELM (He et al., 2026b), we implement the HELM architecture within the NanoChat (Karpathy, 2025) and use the same training and evaluation framework as our method. HELM uses the same aspect ratio, head dimension, and maximum sequence length as the Euclidean baseline and LBLM. At each matched depth, its curvature is set to the same value as LBLM. Unlike the Euclidean baseline and LBLM, HELM is a fully hyperbolic architecture that introduces multiple nonlinear hyperbolic operations throughout the Transformer blocks. Therefore, we do not intentionally restrict HELM by forcing it to use the same optimization settings as LBLM. Instead, we adopt the official HELM optimization settings (He et al., 2026b), which are specifically designed for stable training of the fully hyperbolic architecture. This choice is necessary to ensure stable optimization and a meaningful comparison rather than forcing all architectures to share an unsuitable training configuration. As shown in Figure 7b, when HELM at depth 12 is trained using the same training recipe (*Our Training Recipe*) as the Euclidean baseline and LBLM, the validation BPB fails to converge and exhibits unstable training behavior. We therefore report HELM using its official training recipe (*HELM Official Recipe*).

Training data and evaluation. We pretrain all models on NVIDIA ClimbMix dataset (Diao et al., 2026), using the data loading pipeline provided by NanoChat (Karpathy, 2025). The number of tokens used for each model follows the training budgets described above. We evaluate the pretrained models on seven benchmarks: ARC-Easy, ARC-Challenge (Clark et al., 2018), COPA (Roemmele et al., 2011), CommonsenseQA (Talmor et al., 2019), OpenBookQA (Mihaylov et al., 2018), BoolQ (Clark et al., 2019), and BIG-bench Language Identification (BIG-bench authors, 2023). All evaluations are conducted using the evaluation scripts provided by NanoChat, and we report accuracy on each benchmark.

E.2 Model Configurations and Training Budgets

Table 9 summarizes the model configurations and total training tokens used across different model depths. Our LBLM model and the Euclidean baseline are both built on the standard Transformer. At each depth, we match the model dimension and number of attention heads between the standard Transformer used by LBLM and the Euclidean baseline and HELM. All pretraining runs are conducted on two NVIDIA H100 GPUs with a fixed global batch size, so the device batch size affects hardware efficiency but not optimization. Because of its higher memory footprint, HELM fits a much smaller device batch size than the Euclidean baseline and LBLM. We report the total parameter count of models and the number of scaling parameters used to determine the total training tokens. Using the same ratio of scaling parameters to training data, 1:12, the total number of training tokens increases with model depth.

Table 9: **Model configurations and training budgets across different depths.** Following the official NanoChat configuration, all models use an aspect ratio of 64, a head dimension of 128, and a maximum sequence length of 2048. The number of total training tokens is set to 12 times the number of scaling parameters. All models are trained on NVIDIA H100 GPUs with the same global batch size; differences in device batch size are compensated by gradient accumulation. The device batch size of HELM is limited by GPU memory.

Depth	Dim.	Heads	Euclidean Baseline and LBLM				HELM		
			Total Params. (B)	Scaling Params. (B)	Training Tokens (B)	Device Batch	Scaling Params. (B)	Training Tokens (B)	Device Batch
12	768	6	0.29	0.11	1.32	32	0.14	1.67	8
16	1024	8	0.54	0.24	2.82	32	0.30	3.64	4
20	1280	10	0.90	0.44	5.22	16	0.57	6.83	2
24	1536	12	1.38	0.73	8.76	16	–	–	–

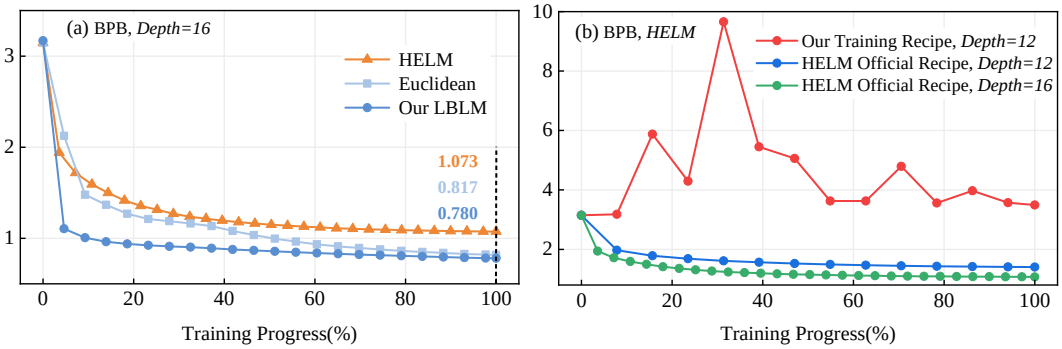


Figure 7: **Validation BPB comparison.** (a) Validation BPB of HELM, the Euclidean baseline, and LBLM at depth 16. (b) Validation BPB of HELM at depths 12 and 16, where *Our Training Recipe* is the same training hyperparameters as Euclidean baseline and our LBLM, the official configuration follows the HELM (He et al., 2026b) training settings.

Following NanoChat (Karpathy, 2025), we define the scaling parameters as the parameters in the Transformer blocks together with those in the output layer. HELM uses the same vocabulary size for the output layer as Euclidean baseline and our LBLM. At each scale, the standard Transformer used by the Euclidean baseline and LBLM is matched to HELM in depth, model dimension, and number of attention heads. However, HELM operates on an $(n+1)$-dimensional Lorentz representation and introduces additional components for hyperbolic computation, resulting in a larger number of scaling parameters (He et al., 2026b). The primary difference comes from the hyperbolic FFN in HELM, which contains more parameters than the corresponding FFN in the standard Transformer. Smaller additional differences arise from the extra Lorentz coordinate, biases in the attention projections, learnable parameters in Lorentz RMSNorm and residual connections, and the additional final projection and normalization layers. Together, these architectural differences account for the larger scaling parameter count of HELM.

F End-to-End Training Efficiency

LBLM replaces the standard Euclidean linear output layer used to compute vocabulary logits. Both the Euclidean and HELM baselines compute logits using a Euclidean linear inner product readout, whereas LBLM replaces this with the LBL. In this section, we describe the implementation of LBLM and analyze how its additional computations are organized to keep the end-to-end training cost close to that of the Euclidean baseline.

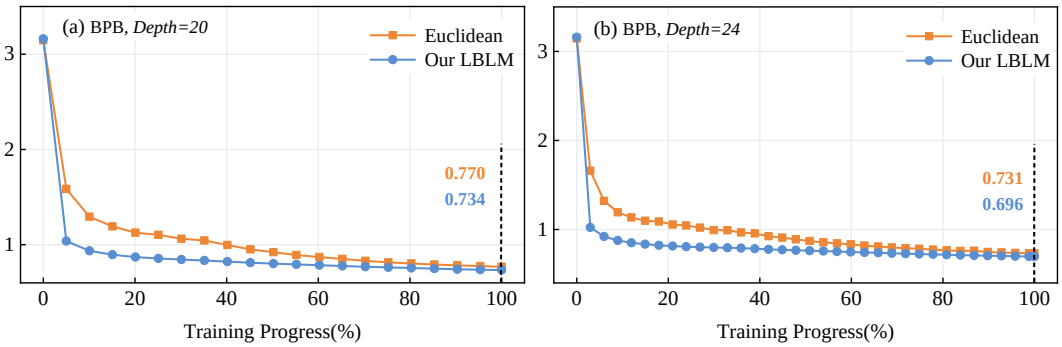


Figure 8: **Validation BPB comparison.** (a) Validation BPB of the Euclidean baseline, and LBLM at depth 20. (b) Validation BPB of the Euclidean baseline, and LBLM at depth 24.

For the end-to-end training time comparison, we focus on the Euclidean baseline and LBLM, since they use the same GPT backbone and the same training token budget at each depth. As discussed in Appendix E.2, HELM contains more parameters at the same depth and is therefore trained with a larger token budget, making its total training time not directly comparable under the same training workload. Nevertheless, to show the practical training time difference between HELM and our method under the same depth, number of heads, and model dimension, we also report HELM training time.

F.1 Forward Computation

Readout computation. The LBL in Eq. 2 retains the main matrix multiplication used by a standard Euclidean output layer. Let N denote the number of token positions processed in one output-layer call, $|\mathcal{V}|$ the vocabulary size, and n the hidden dimension. Let $\mathbf{H} \in \mathbb{R}^{N\times n}$ collect the hidden states, and let $\mathbf{W} \in \mathbb{R}^{|\mathcal{V}|\times n}$ collect the spatial coordinates $\mathbf{w}_j$ of the vocabulary embeddings. Writing $R = 1/\sqrt{c}$, we first compute

$$\mathbf{u}_i = \frac{\mathbf{h}_i}{\|\mathbf{h}_i\|}, \qquad x_{j,0} = \sqrt{R^2 + \|\mathbf{w}_j\|^2}. \tag{102}$$

If $\mathbf{U}$ collects the normalized hidden states, the logits become

$$\mathbf{S} = \mathbf{U}\mathbf{W}^\top, \qquad \ell_{ij} = -\alpha R \log\left(\frac{x_{j,0} - S_{ij}}{R}\right). \tag{103}$$

The only forward operation that combines every hidden state with every vocabulary embedding across the hidden dimension is $\mathbf{U}\mathbf{W}^\top$. It has the same matrix dimensions and $\Theta(N|\mathcal{V}|n)$ arithmetic cost as the matrix multiplication $\mathbf{H}\mathbf{W}^\top$ in the standard Euclidean output layer. Both matrix multiplications can be performed in BF16 using NVIDIA H100 Tensor Cores.

Hidden state normalization and reconstruction of the time coordinates require $\Theta(Nn)$ and $\Theta(|\mathcal{V}|n)$ operations, respectively. Both are computed separately from the $\mathbf{U}\mathbf{W}^\top$ matrix multiplication: each normalized hidden state is reused across all vocabulary entries, while each reconstructed time coordinate is reused across all token positions. The implementation stores the spatial coordinates and computes the time coordinates separately, avoiding construction of a complete $|\mathcal{V}| \times (n+1)$ Lorentz embedding.

Fused loss computation. During training, after computing $\mathbf{S} = \mathbf{U}\mathbf{W}^\top$, a fused Triton kernel (Tillet et al., 2019) combines the Busemann function with cross entropy loss computation. Specifically, the subtraction, division, logarithm, and scaling in LBL are fused with the log-sum-exp reduction over the vocabulary and the extraction of the target logit.

The pairwise dot products use a temporary BF16 buffer. The geometric operations and loss reductions are evaluated in FP32 within the fused computation, without materializing a complete FP32

Table 10: **End-to-end training time in hours.** Results are averaged over three runs, except for HELM at depth 20; all experiments are conducted on NVIDIA H100 GPUs. At each depth, the Euclidean baseline and LBLM use the same GPT backbone and training configuration, with their respective output layers and optimizers.

Method	Depth 12	Depth 16	Depth 20	Depth 24
HELM	6.439	26.632	89.732	–
Euclidean	0.626	2.302	7.577	18.237
LBLM	0.657	2.368	7.568	18.489

logit tensor of shape $N \times |\mathcal{V}|$. This avoids storing and repeatedly reading full FP32 intermediate tensors between these operations. The BF16 matrix multiplication buffer still requires storage proportional to $N|\mathcal{V}|$.

Baseline implementation. To ensure that the timing comparison reflects the cost of the output layer rather than differences in implementation effort, the Euclidean baseline computes the logits $\mathbf{H}\mathbf{W}^\top$ with a BF16 matrix multiplication and evaluates the loss with a fused Triton cross entropy kernel that combines the log-sum-exp reduction over the vocabulary with the extraction of the target logit. LBLM follows the same structure, so the two implementations differ only in the operations specific to logits calculation.

F.2 Backward Computation and Training Time

Backward computation. We implement the backward pass using a custom autograd function. It recomputes the BF16 matrix multiplication and combines the softmax and Busemann function in fused Triton kernels. Two further matrix multiplications compute the gradient contributions through $\mathbf{S}$ to $\mathbf{U}$ and $\mathbf{W}$. Differentiation through hidden state normalization and time coordinate reconstruction supplies the remaining gradient contributions. Gradients of the shared scale α and radius R are accumulated using reductions with a fixed summation order, without atomic additions. The radius gradient is subsequently propagated to the learnable curvature parameter.

Including the initial $\mathbf{U}\mathbf{W}^\top$ computation in the forward pass, the implementation performs four large matrix multiplications across the forward and backward passes. Recomputing $\mathbf{U}\mathbf{W}^\top$ during the backward pass introduces an additional $\Theta(N|\mathcal{V}|n)$ operation relative to an implementation that retains the required intermediate results. This additional computation is included in the end-to-end training times reported below.

For the forward computation, after hidden state normalization and the $\mathbf{U}\mathbf{W}^\top$ matrix multiplication, LBL introduces only $\Theta(N|\mathcal{V}|)$ additional elementwise operations, while the shared matrix multiplication requires $\Theta(N|\mathcal{V}|n)$ computation. As the hidden dimension n increases, the matrix multiplication therefore accounts for a larger fraction of the total forward computation, reducing the relative overhead introduced by LBL. This behavior is consistent with Figure 6b, where the throughput gap between the Euclidean inner product logit and our LBL narrows as n increases when explicitly computing the full vocabulary logits.

Training time. Table 10 reports the end-to-end training time on NVIDIA H100, averaged over three runs with random seeds $\{41, 42, 43\}$, except for HELM at depth 20, which was run once. For the controlled comparison between the Euclidean baseline and LBLM, both methods use the same GPT backbone, shared training configuration, and training-token budget at each depth, as reported in Table 9.

Across all evaluated depths, LBLM closely matches the training time of the Euclidean baseline. The relative overhead is small and becomes particularly limited at larger depths, where the training times of the two methods are nearly identical. These results indicate that the additional computations in LBL and CRAdam updates introduce only a modest end-to-end training overhead. This is consistent with the implementation described above, where LBLM retains the dominant matrix multiplication of the standard Euclidean output layer and adds mainly lower cost elementwise operations.

Table 11: **Recipe controlled comparison at depth 12.** Both HELM and the Euclidean baseline are trained using the official HELM training recipe. Results are averaged over 3 runs. Best results are in **bold**.

Method	ARC-E	ARC-C	COPA	CSQA	OBQA	BoolQ	BB-Lang	Avg. (↑)
HELM	31.4	**22.4**	**47.3**	**31.2**	**26.0**	**44.3**	24.9	**32.53** ± 1.39
Euclidean	**33.9**	21.3	37.7	27.7	24.5	40.7	**25.4**	30.17 ± 0.38

G Comparison with HELM

The main results include HELM (He et al., 2026b) as a hyperbolic baseline alongside the Euclidean baseline. This appendix describes how this comparison is set up and analyzes it in terms of accuracy, training cost, and inference efficiency.

Implementation and training recipe. We implement HELM and match its depth, model dimension, number of attention heads, and token allocation rule to those of the Euclidean baseline and LBLM. As shown in Table 9, because the Lorentz layers of HELM carry additional parameters, HELM has more scaling parameters than the other two models at every depth and therefore, under the shared allocation rule, receives more training tokens. HELM departs substantially from a Euclidean Transformer: its attention is computed from negative squared Lorentz distances $d_c^2(\boldsymbol{x}, \boldsymbol{y})$ rather than dot products, and its normalization, residual connections, positional encoding, and feed-forward networks are all defined through geometric operations in Lorentz model (He et al., 2026b). Stable training of HELM architecture depends on settings that its authors established for it, and a training recipe developed for a Euclidean backbone is not a neutral choice. Indeed, as shown in Figure 7b, HELM does not train stably under the recipe we use for the Euclidean baseline and LBLM: its validation BPB fails to decrease and the run becomes unstable. We therefore train HELM with the optimization settings released by He et al. (2026b). This follows the usual practice of comparing complete methods under their respective recommended configurations. As a consequence, comparisons between LBLM and HELM reflect differences in both method and recipe.

Reimplementation check. To verify our HELM reimplementation, we also train a Euclidean backbone with the recipe of He et al. (2026b) at depth 12 and evaluate both models under our protocol. As shown in Table 2 (per-task results in Table 11), HELM reaches 32.53 against 30.17 for this backbone, reproducing the direction of the improvement reported by He et al. (2026b). Since this check concerns the implementation rather than the scaling behavior of HELM, we perform it at depth 12 only. The same comparison also shows that the training recipe strongly affects absolute accuracy: the Euclidean backbone reaches 30.17 under the recipe of He et al. (2026b), compared with 38.75 under ours.

Accuracy. We compare LBLM and HELM as complete methods, each trained with the recipe recommended for it, which reflects how a practitioner would adopt either method. As shown in Table 12, LBLM achieves higher average accuracy than HELM at all three evaluated depths, even though HELM receives more training tokens. As discussed above, this comparison reflects method and recipe jointly, so we do not interpret it as evidence that one architecture is intrinsically more effective than the other. It does, however, highlight a practical advantage of LBLM. Since LBLM modifies only the output layer, it trains stably under our recipe, whereas HELM relies on its own recipe, under which the Euclidean backbone attains substantially lower accuracy in our setting, as the reimplementation check shows.

Cost of the improvement. As shown in Table 2, which reports at depth 12 each method alongside a Euclidean backbone trained under the same recipe, together with the number of training tokens and the training time of every model, both methods improve over their baselines, but at very different costs. LBLM improves over the Euclidean baseline (38.75) by 3.03 points with the same training budget of 1.32B tokens and only 5% additional training time; detailed timing results are reported in Table 10. HELM improves over its baseline by 2.36 points, but with more scaling parameters (0.14B versus 0.11B), 1.27× as many training tokens (1.67B versus 1.32B), and about 10× the

training time of that baseline, whose training time is 0.672 hours. This timing result is reported here but is not included in Table 10 for completeness. Its improvement is therefore not obtained at a matched budget, and we cannot rule out that part of it stems from the larger budget rather than from the architecture. Moreover, even with this improvement, HELM (32.53) remains well below both the Euclidean baseline trained with our recipe (38.75) and LBLM (41.78).

Training efficiency. Beyond accuracy, the two methods differ substantially in computational cost. LBLM confines hyperbolic geometry to the output layer and retains the dominant matrix multiplication of a standard Euclidean output layer, whereas HELM executes hyperbolic operations at every layer. To compare the two architectures under an identical workload, we measure training throughput with LBLM run at the device batch size of HELM. As shown in Figure 4, under this setting the training throughput of HELM is 72, 38, and 21 thousand tokens per second at depths 12, 16, and 20, compared with 455, 241, and 112 thousand for LBLM, a gap of 5.3–6.3×. This per-token gap is independent of how many tokens each method is allocated. It is also conservative for LBLM, which in actual training uses a 4–8× larger device batch size and attains higher throughput than reported here.

End-to-end training time offers a complementary view but must be interpreted with care. HELM requires about 6.4, 26.6, and 89.7 hours of end-to-end training at depths 12, 16, and 20. As shown in Table 9, however, HELM is trained with more tokens and a smaller device batch size than LBLM, so these times are not directly comparable with those of LBLM, and we do not use them to compare the two methods. The relevant end-to-end comparison for LBLM is instead with the Euclidean baseline, which shares its recipe, token budget, and device batch size. As shown in Table 10, LBLM trains in nearly the same time as the Euclidean baseline: 0.657 versus 0.626, 2.368 versus 2.302, 7.568 versus 7.577, and 18.489 versus 18.237 hours at depths 12, 16, 20, and 24. The overhead is about 5% at depth 12 and within 2% at depths 20 and 24.

Inference efficiency. We further compare autoregressive inference throughput. All measurements are conducted on a single NVIDIA H100 80GB GPU with a batch size of 1. This setting reflects the latency of a single user interacting with the model, rather than the throughput of batched serving. Each run uses a deterministically generated random prompt of fixed length 128. The same prompt seeds are used across models, while the five measured runs use different prompts. All models decode with key–value (KV) cache. The prompt is first processed in a single prefill pass, which fills the cache and produces the first output token. Each of the remaining 1919 decoding steps then takes only the most recently generated token as input, rather than recomputing the full context. Throughput is defined as the total number of generated tokens, i.e., the batch size multiplied by the 1920 output tokens per sequence, divided by the total GPU time spent in the prefill and decoding phases. It therefore includes the cost of prefill. It excludes model loading, the creation of the prompt tensor, the allocation of the KV cache, and the warm-up runs. For each configuration, we perform two warm up runs, which are discarded, followed by five measured runs. We report the median throughput over the five measured runs.

The throughput gap observed in training carries over to inference. As shown in Figure 3c, under identical decoding settings LBLM reaches 105, 87, and 73 tokens per second at depths 12, 16, and 20, which is 2.8–3.2× the 38, 28, and 23 tokens per second of HELM. As further shown in Figure 3c, LBLM retains 79%, 83%, and 88% of the throughput of the Euclidean baseline, whereas HELM retains less than 30%. These results are consistent with the architectural difference between the two methods. HELM performs hyperbolic computations throughout the Transformer backbone, so these operations are repeated at every autoregressive decoding step. In contrast, LBLM retains the standard Euclidean backbone and introduces additional geometric computation only when producing the vocabulary logits from the final hidden state. For the same reason, the relative overhead of LBLM decreases with depth, since the cost of its output layer grows more slowly than that of the backbone as the model deepens.

In summary, LBLM attains higher accuracy than HELM when each is trained with its own recipe, and it obtains its improvement over the Euclidean baseline at essentially no additional training time. Its training time and token throughput are comparable to those of the Euclidean baseline, while its token throughput is substantially higher and its training time much lower than those of HELM.

Table 12: **Pretraining results of LBLM and HELM.** Results are averaged over 3 runs, except for HELM at depth 20. Best results are in **bold**.

Method	Task Performance							Metric
	ARC-E	ARC-C	COPA	CSQA	OBQA	BoolQ	BB-Lang	Avg. (↑)
Depth 12								
HELM	31.4	22.4	47.3	31.2	26.0	44.3	24.9	32.53 ± 1.39
LBLM (ours)	**56.9**	**28.6**	**59.0**	**34.5**	**31.9**	**55.9**	**25.7**	**41.78** ± 0.35
Depth 16								
HELM	38.2	23.7	45.7	27.7	25.9	49.2	25.6	33.70 ± 1.55
LBLM (ours)	**62.3**	**33.0**	**62.0**	**28.0**	**35.5**	**59.1**	**25.9**	**43.69** ± 0.17
Depth 20								
HELM	42.2	23.5	46.0	**27.9**	27.4	51.5	**25.4**	34.84
LBLM (ours)	**66.5**	**36.5**	**66.7**	26.4	**38.6**	**55.8**	25.3	**45.11** ± 0.54

H Experimental Analysis

H.1 Comparison with BMLR

BMLR implementation. We implement the BMLR formulation of Chen et al. (2026) for vocabulary logits, treating each vocabulary token as a class. Given a Transformer hidden state $\mathbf{h} \in \mathbb{R}^n$, we apply the exponential map at the Lorentz pole:

$$\boldsymbol{z}_{\mathbf{h}} = \exp_{\boldsymbol{o}}^{c}\left([0, \phi(\mathbf{h})]^\top\right) \in \mathbb{L}_c^n, \tag{104}$$

where n is the hidden dimension, $c > 0$ is the curvature magnitude, and $\boldsymbol{o}$ is the pole of the Lorentz model. Moreover, $\phi(\mathbf{h})$ is a transformation applied to the hidden state before the exponential map, with $\phi(\mathbf{h}) = \mathbf{h}$ unless otherwise stated; the choices of ϕ are examined below.

Let $\mathbf{w}_j \in \mathbb{R}^n$ denote the stored vocabulary parameter for token j. For $\mathbf{w}_j \neq \mathbf{0}$, we construct

$$\boldsymbol{\xi}_j = [1, \mathbf{w}_j / \|\mathbf{w}_j\|]^\top. \tag{105}$$

In the original BMLR formulation, each vocabulary token has its own learnable scale α_j and bias e_j. Using the Busemann notation adopted in this work, the resulting logit is

$$\ell_j^{\mathrm{BMLR}}(\mathbf{h}) = -\alpha_j b_{\boldsymbol{\xi}_j}(\boldsymbol{z}_{\mathbf{h}}) + e_j. \tag{106}$$

Thus, the boundary direction, scale, and bias are parameterized separately for each vocabulary token.

For a parameter-matched comparison with LBLM, we additionally consider BMLR-PM, which removes the class-specific scale α_j and bias e_j and instead uses a single learnable scale α shared across all vocabulary tokens:

$$\ell_j^{\text{BMLR-PM}}(\mathbf{h}) = -\alpha b_{\boldsymbol{\xi}_j}(\boldsymbol{z}_{\mathbf{h}}). \tag{107}$$

BMLR-PM therefore retains the BMLR boundary construction and hidden state exponential mapping while matching the parameterization of LBLM by using only a shared logit scale and no class-specific bias. We include BMLR-PM to provide a parameter matched comparison with LBLM and to isolate the effect of the BMLR logit formulation from the additional class specific scale and bias parameters used in the original BMLR.

Training implementation. In the BMLR comparison, we replace the LBL logit formulation in LBLM with BMLR while using the same training configurations and hyperparameters reported in Appendix E. During training, LBLM and BMLR share the same Triton based fused implementation for cross entropy loss computation. The pairwise dot products are stored in a temporary BF16 buffer, while the geometric operations and loss reductions are evaluated in FP32 without storing a complete FP32 vocabulary logit tensor. The fused loss computation used during training is described separately in Appendix F on end-to-end training efficiency.

Table 13: **Comparison with BMLR under different hidden state scaling strategies.** Here, $\phi(\mathbf{h})$ denotes the transformation applied to the hidden state before it is mapped to hyperbolic space by the exponential map, after which the resulting hyperbolic hidden state is used as the input to BMLR. We compare full BMLR and BMLR-PM using three choices of $\phi(\mathbf{h})$: the original hidden state $\mathbf{h}$, the normalized hidden state $\mathbf{h}/\|\mathbf{h}\|$, and the hidden state scaled by $1/\sqrt{n}$. For each method, we then evaluate its best configuration of $\phi(\mathbf{h})$ over three random seeds. The final block reports the corresponding mean results, with the standard deviation of the average score. Best results within each block are shown in **bold**.

Method / $\phi(\mathbf{h})$	**Task Performance**							**Avg. (↑)**
	ARC-E	**ARC-C**	**COPA**	**CSQA**	**OBQA**	**BoolQ**	**BB-Lang**	
BMLR-PM (single seed scaling comparison)								
BMLR-PM ($\mathbf{h}$)	32.6	23.7	45.0	20.3	25.8	**49.6**	**25.2**	31.74
BMLR-PM ($\mathbf{h}/\|\mathbf{h}\|$)	54.3	26.9	**58.0**	22.4	32.0	47.3	24.7	37.94
BMLR-PM ($\mathbf{h}/\sqrt{n}$)	**54.8**	**27.5**	56.0	**32.1**	**32.2**	48.9	24.9	**39.49**
BMLR (single seed scaling comparison)								
BMLR ($\mathbf{h}$)	48.3	26.4	48.0	**35.4**	28.4	**56.1**	25.4	38.29
BMLR ($\mathbf{h}/\|\mathbf{h}\|$)	**56.9**	27.4	**56.0**	25.9	**32.8**	53.2	25.3	**39.64**
BMLR ($\mathbf{h}/\sqrt{n}$)	56.2	**28.2**	54.0	26.0	32.6	47.0	**25.6**	38.51
Three seed comparison								
BMLR-PM ($\mathbf{h}/\sqrt{n}$)	54.9	27.5	56.3	30.1	**32.5**	53.4	25.1	39.98 $\pm$ 0.67
BMLR ($\mathbf{h}/\|\mathbf{h}\|$)	55.9	26.8	56.0	26.5	32.1	54.6	25.1	39.57 $\pm$ 0.55
Ours	**56.9**	**28.6**	**59.0**	**34.5**	31.9	**55.9**	**25.7**	**41.78** $\pm$ 0.35

Effect of hidden state scaling for BMLR. Since the original BMLR formulation constrains the input magnitude, we further evaluate several transformations of $\mathbf{h}$ before the exponential map for a more complete comparison. Specifically, we consider $\mathbf{h}$, $\mathbf{h}/\|\mathbf{h}\|$, and $\mathbf{h}/\sqrt{n}$, each of which is mapped to hyperbolic space via the exponential map before being passed to BMLR. These are first compared with random seed 43 as shown in Table 13; consequently, the single-seed results for BMLR ($\mathbf{h}$) and BMLR-PM ($\mathbf{h}$) differ slightly from the averages in Table 4. Both variants are sensitive to the choice of scaling method of $\mathbf{h}$, and the unscaled hidden state yields the lowest average accuracy for each. We then select the transformation with the highest single-seed average for each variant, $\mathbf{h}/\sqrt{n}$ for BMLR-PM and $\mathbf{h}/\|\mathbf{h}\|$ for BMLR, and evaluate it over three seeds (Table 4). Over three random seeds, appropriate scaling improves BMLR-PM from 32.24 to 39.98 and BMLR from 36.96 to 39.57. In contrast, LBL depends on $\mathbf{h}$ only through its direction $\mathbf{u}$, so its logits are invariant to $\|\mathbf{h}\|$; it achieves the highest average accuracy without requiring any choice of hidden state scaling.

Results. Table 4 reports results for BMLR and BMLR-PM under three random seeds $\{41, 42, 43\}$, both with the unscaled hidden state and with the scaling transformation $\phi(\mathbf{h})$ selected above. LBL achieves the highest average accuracy among all configurations, 41.78 ± 0.35. Even against the best-scaled variants, it exceeds BMLR-PM (39.98 ± 0.67) by 1.80 points and BMLR (39.57 ± 0.55) by 2.21 points, and it is higher than both on six of the seven tasks, with OpenBookQA as the exception. We do not claim that these three choices exhaust the possible transformations, and a better one may exist for BMLR. This is itself a difference between the two formulations: the accuracy of BMLR depends on a transformation that has to be selected, and potentially re-selected for each model, whereas the logits of LBL are invariant to $\|\mathbf{h}\|$ and involve no such choice. LBL also obtains these results with a simpler output layer. BMLR introduces a class-specific scale α_j and bias e_j for every vocabulary token, i.e., $2|\mathcal{V}|$ additional parameters, whereas LBL uses only a single shared scale α and still outperforms it; LBL also outperforms the parameter matched BMLR-PM, so its advantage does not stem from parameter count. Moreover, the two kinds of logits also differ in how they use the Transformer hidden states and vocabulary representations. BMLR maps each hidden state to a hyperbolic point and uses token-specific boundary directions constructed from

normalized vocabulary weights. In contrast, LBL uses the normalized hidden state itself as the boundary direction and evaluates the resulting Busemann function over learnable hyperbolic token embeddings. This construction avoids both the exponential mapping of hidden states and the full vocabulary weight normalization required by BMLR.

H.1.1 Inference Efficiency

We compare the computational efficiency of three logits: the standard Euclidean linear inner product, BMLR, and our proposed LBL. Both BMLR and LBL are implemented directly according to their defining equations, without method-specific algebraic reformulation or optimization. The results therefore compare the two formulations as written, rather than the most efficient implementation attainable for each; optimized implementations of BMLR, for example ones that avoid materializing the normalized vocabulary matrix, could reduce its overhead, which is beyond the scope of this work. All experiments are conducted on a single NVIDIA H100 GPU. To isolate the computational cost of the output layer, we exclude the Transformer backbone and directly feed the same synthetic final hidden states to each LM head. For all experiments, we fix the number of input hidden states to 2048 and the vocabulary size to 131072, while varying the hidden size over $\{1024, 2048, 4096, 6144, 8192\}$. The same randomly generated hidden states are used for the three methods at each hidden size, and BF16 is used for the input and matrix-multiplication precision. For this output layer benchmark, each method explicitly constructs the complete vocabulary logits for all input hidden states.

Output Layer Throughput. For each configuration, we first perform 10 untimed warm-up forward passes, followed by 50 measured forward passes. The execution time of each forward pass is measured using CUDA events, and the GPU is synchronized before the recorded timings are collected. We report the mean latency across the measured iterations and convert it into throughput according to the number of input token representations processed per second.

The Euclidean output layer provides a particularly efficient reference point on the NVIDIA H100 GPU. Its logit computation consists primarily of a single linear projection between the hidden state matrix and the vocabulary weight matrix $\mathbf{W} \in \mathbb{R}^{|\mathcal{V}|\times n}$. Because the hidden states are stored in BF16, the vocabulary weights are cast to the same dtype before the projection, such that the dominant matrix multiplication is performed in BF16. This operation can directly benefit from the high throughput BF16 matrix multiplication kernels provided by the H100 Tensor Cores.

In contrast, both hyperbolic output layers including BMLR and our LBL require additional geometric operations beyond the main pairwise projection. These computations involve FP32 arithmetic in our implementation for numerically sensitive operations. Such operations cannot exploit the same highly optimized BF16 matrix multiplication path as the Euclidean projection and introduce additional computation and memory traffic. Consequently, the Euclidean output layer concentrates most of its computational cost in a single BF16 matrix multiplication, whereas the hyperbolic output layers including BMLR and LBLM additionally execute a sequence of FP32 geometric operations. This difference contributes to the substantially higher throughput of the Euclidean baseline observed in Figure 6b. Since this benchmark isolates the output layer, it does not reflect end-to-end cost. In end-to-end measurements, LBL retains 79–88% of the Euclidean baseline's inference throughput (Figure 3c), and in training, where it uses the fused kernel described in Appendix F, its overhead is at most about 5% (Table 10).

Under these direct implementations, our LBL achieves higher throughput than BMLR at all evaluated hidden sizes, and the gap widens as n grows. The difference arises from which side of the logit computation is normalized. In BMLR, the boundary directions are the normalized vocabulary vectors $\mathbf{w}_j/\|\mathbf{w}_j\|$, so a direct implementation normalizes the full matrix $\mathbf{W} \in \mathbb{R}^{|\mathcal{V}|\times n}$ in every forward pass and additionally maps each hidden state with the exponential map. In LBL, the boundary directions come from the hidden states, so only the N hidden states are normalized and the unnormalized $\mathbf{W}$ enters the matrix product directly; the vocabulary side contributes only the time coordinates $x_{j,0}$, a vector of length $|\mathcal{V}|$. LBL thus attains its efficiency in its direct form, whereas bringing BMLR to a comparable cost would require reformulating its computation.

Peak additional CUDA memory. GPU memory is measured separately from the latency benchmark. Before the memory measurement, temporary objects are released, the CUDA cache is cleared,

Table 14: **Comparison of output layer configurations.** Detailed definitions of output layer configurations are provided in Table 15. For Variant IV, the two settings differ only in whether the softcap is applied. Best results are in **bold**.

Variant	Task Performance							Avg. (↑)
	ARC-E	ARC-C	COPA	CSQA	OBQA	BoolQ	BB-Lang	
I	52.7	26.4	54.0	34.3	28.6	51.1	25.5	38.94
II	53.9	27.5	56.0	33.4	30.2	45.7	25.1	38.83
III	36.1	22.0	44.0	36.0	26.2	**60.9**	25.5	35.81
IV	53.9	27.5	54.0	31.1	**32.2**	52.8	25.4	39.56
IV (w/ softcap)	24.2	22.7	51.0	19.6	27.4	37.8	25.0	29.67
Ours	**57.6**	**28.7**	**61.0**	34.3	32.0	56.4	25.7	**42.24**

Table 15: **Definitions of output layer configurations.**

Variant	Configuration	Logit Formulation	Space
I	Euclidean inner product logits	$\mathbf{h}^\top \mathbf{w}_j$	$\mathbb{R}^n$
II	Euclidean inner product + learnable scale α	$\alpha \mathbf{h}^\top \mathbf{w}_j$	$\mathbb{R}^n$
III	Euclidean geodesic distance logits	$-\lVert \mathbf{h} - \mathbf{w}_j \rVert$	$\mathbb{R}^n$
IV	LBLM with hyperbolic geodesic distance logits	$-\alpha d_{\mathbb{L}}^c \left(\exp_{\boldsymbol{o}}^c \left([0, \mathbf{h}]^\top \right), \boldsymbol{x}_j \right)$	$\mathbb{L}_c^n$
Ours	**LBLM with Lorentz Busemann logits**	$-\alpha b_{\boldsymbol{\xi}_\mathbf{h}} (\boldsymbol{x}_j)$	$\mathbb{L}_c^n$

and the GPU is synchronized. We then record the amount of CUDA memory already allocated before the output layer forward pass and reset the CUDA peak memory statistics. After executing one full vocabulary forward pass, we record the maximum CUDA memory allocated during the computation. The value reported in Figure 6a, denoted as *Peak Additional CUDA Memory*, is the increase from the pre-forward allocated memory to this peak value. Because the input hidden states and output layer parameters have already been allocated before the memory baseline is recorded, this measurement excludes their existing storage. Instead, it captures the additional runtime memory required by the output layer computation, including the full logit tensor and method specific intermediate tensors. Consequently, this metric should not be interpreted as the total GPU memory occupied by the model.

As shown in Figure 6a, across the evaluated hidden dimensions, our LBL's additional CUDA memory increased only modestly, whereas BMLR exhibited approximately linear growth because its logit formulation uses the normalization of each vocabulary weight vector as the boundary directions in Busemann function. Computing logits for all N tokens therefore requires normalization of the vocabulary weight matrix $\mathbf{W} \in \mathbb{R}^{|\mathcal{V}| \times n}$, materializing an additional normalized vocabulary matrix $\widehat{\mathbf{W}} \in \mathbb{R}^{|\mathcal{V}| \times n}$ during the forward pass. In contrast, LBLM does not require such normalization on vocabulary weight matrix $\mathbf{W} \in \mathbb{R}^{|\mathcal{V}| \times n}$. It only normalizes the hidden states to obtain boundary directions, so the additional CUDA memory dependent intermediate tensors are much smaller than a full matrix $\mathbf{W} \in \mathbb{R}^{|\mathcal{V}| \times n}$. As a result, its additional CUDA memory grows only modestly with the hidden dimension. LBLM also avoids explicitly mapping the hidden states into hyperbolic space through an exponential map, further reducing intermediate computation and storage.

H.2 Comparison with Alternative Output Layer Configurations

We compare LBLM with alternative output layer configurations in Table 14 to examine whether adding a learnable scale α to Euclidean inner product logits or using negative Euclidean geodesic distance logits can match LBLM's performance. We also evaluate whether replacing LBL within LBLM with negative hyperbolic geodesic distance logits improves performance.

Table 16: Detailed comparison of Bdeir-RAdam and our CRAdam across different logit formulations. BMLR in this comparison uses the hidden state $\mathbf{h}$. All results are obtained with random seed 43.

Logits	Optimizer	Task Performance							Avg. (↑)
		ARC-E	ARC-C	COPA	CSQA	OBQA	BoolQ	BB-Lang	
BMLR	Bdeir-RAdam	34.6	23.7	46.0	22.6	24.4	**57.0**	**25.7**	33.43
	CRAdam	**48.3**	**26.4**	**48.0**	**35.4**	**28.4**	56.1	25.4	**38.29**
Variant IV	Bdeir-RAdam	**54.9**	**27.6**	**55.0**	22.7	31.0	**54.0**	25.3	38.64
	CRAdam	53.9	27.5	54.0	**31.1**	**32.2**	52.8	**25.4**	**39.56**
LBL	Bdeir-RAdam	55.1	27.1	56.0	33.3	30.0	54.5	25.2	40.17
	CRAdam	**57.6**	**28.7**	**61.0**	**34.3**	**32.0**	**56.4**	**25.7**	**42.24**

Specifically, we first augment the standard Euclidean inner product logits with the same learnable scale used in LBLM (variant II), providing a controlled comparison that separates the effect of the scaling factor α from that of the LBL. We then compare the proposed LBL with logits based on negative Euclidean geodesic distance (variant III) and negative hyperbolic geodesic distance (variant IV) to determine whether these geodesic distance alternatives provide better vocabulary prediction performance than Busemann function. The hyperbolic variant IV replaces LBL within LBLM with negative geodesic distance logits. We compare LBLM with distance based logit formulations in Euclidean and hyperbolic spaces.

Let $\mathbf{h} \in \mathbb{R}^n$ denote a Transformer hidden state and $\mathbf{w}_j \in \mathbb{R}^n$ the output embedding of vocabulary token j, where n is the hidden dimension. Variant I uses the standard Euclidean inner product (IP):

$$\ell_j^{\mathrm{IP}}(\mathbf{h}) = \mathbf{h}^\top \mathbf{w}_j. \tag{108}$$

Variant II introduces a shared learnable logit scale α, giving a scaled inner product (SIP):

$$\ell_j^{\mathrm{SIP}}(\mathbf{h}) = \alpha \mathbf{h}^\top \mathbf{w}_j. \tag{109}$$

Variant III computes logits using negative Euclidean distance:

$$\ell_j^{\mathrm{ED}}(\mathbf{h}) = -\|\mathbf{h} - \mathbf{w}_j\|. \tag{110}$$

The hyperbolic variant IV replaces the LBL in LBLM with negative hyperbolic geodesic distance logits. The hidden state is mapped to a Lorentz point through the exponential map:

$$\boldsymbol{z}_{\mathbf{h}} = \exp_{\boldsymbol{o}}^c\left([0, \mathbf{h}]^\top\right) \in \mathbb{L}_c^n, \tag{111}$$

where $\boldsymbol{o}$ is the Lorentz origin and $c > 0$ is the curvature magnitude. Since the hyperbolic geodesic distance logits operate on a substantially different numerical scale from LBL, applying the same softcap imposes a much stronger constraint on their logit range. We therefore evaluate Variant IV both with and without the softcap while keeping all other training configurations identical. As shown in Table 14, applying the softcap substantially reduces its performance; we therefore use the version without the softcap as Variant IV in the main comparison. Given the learnable Lorentz token embedding $\boldsymbol{x}_j \in \mathbb{L}_c^n$, the logit is

$$\ell_j^{\mathrm{GD}}(\mathbf{h}) = -\alpha d_{\mathbb{L}}^c\left(\boldsymbol{z}_{\mathbf{h}}, \boldsymbol{x}_j\right), \tag{112}$$

where $d_{\mathbb{L}}^c$ denotes the Lorentz geodesic distance and α is a shared learnable logit scale.

Adding a learnable logit scale α does not improve the Euclidean inner product logits, and it remains below LBLM. Both the Euclidean and hyperbolic geodesic distance variants also underperform LBLM. Overall, LBLM achieves the best average performance among the evaluated configurations.

H.3 Riemannian Optimization

This section analyzes the optimization of LBLM. Appendix H.3.1 ablates the components of CRAdam, Appendix H.3.2 compares CRAdam with Bdeir-RAdam, and Appendix H.3.3 compares their computational efficiency.

H.3.1 Ablation of CRAdam

Table 6 ablates the components of CRAdam. The Adam variants are evaluated with a single run, and the CRAdam variants are averaged over three random seeds $\{41, 42, 43\}$. Each variant is compared with the complete CRAdam; the Bdeir-RAdam row, also averaged over three seeds, is discussed separately in Appendix H.3.2.

Learnable curvature and Riemannian optimization. We first compare fixed and learnable curvature under standard Adam. When the curvature is updated, we preserve the spatial coordinates of the Lorentz token embeddings and reconstruct their time coordinates under the updated curvature, as in CRAdam, described in Appendix D.3. Learning the curvature in this way improves the average score from 36.19 to 37.69, with gains on six of the seven tasks, suggesting that learnable curvature is beneficial even without Riemannian optimization. Riemannian optimization brings a larger improvement. Under fixed curvature, CRAdam requires no curvature transition and reduces to the fixed curvature Riemannian Adam update in Appendix D; it raises the average score from 36.19 to 40.75 over Adam. Under learnable curvature, replacing Adam with CRAdam w/o MT, which uses the same point transition and likewise leaves the first moment unchanged, raises the average score from 37.69 to 40.40.

Why LBL calls for Riemannian optimization. The lower scores of the Adam variants should not be interpreted as a limitation of LBL. LBL defines the logits through the hyperbolic geometry of $\mathbb{L}_c^n$, so the effect of a parameter update on the logits is governed by the Lorentz metric rather than by the Euclidean geometry of the stored spatial coordinates. Although the spatial parameterization keeps every token embedding on the manifold under any optimizer, Adam updates $\mathbf{W}$ as if it were Euclidean. Euclidean updates correspond to different geodesic displacements depending on a token's distance from the origin, making Adam's updates inconsistent with the Lorentz geometry. In contrast, CRAdam computes its adaptive direction from Riemannian gradients (Appendix D), consistent with the standard practice of training hyperbolic embeddings with Riemannian optimization (Nickel & Kiela, 2018; Bécigneul & Ganea, 2019). Empirically, replacing Adam with CRAdam at fixed curvature improves the average score, more than learning the curvature under Adam, as shown in Table 6. We therefore attribute the lower scores under Adam to the mismatch between Euclidean updates and the hyperbolic geometry of LBL. LBL and CRAdam are complementary components of LBLM: its gains arise from combining the hyperbolic logits with an optimizer that respects their geometry.

First moment transition. CRAdam w/o MT keeps the first moment unchanged during curvature updates, i.e., $\mathbf{m}_{t+1} = \widetilde{\mathbf{m}}_{t+1}$, while all other components are identical to CRAdam. As shown in Eq. 95, the curvature update alone then changes the metric norm of the radial component of the first moment and thus rescales the subsequent adaptive step. Since the first moment transition reduces to the identity under fixed curvature ($\kappa_t = 1$), its contribution can only be assessed with learnable curvature; the ablation therefore removes the first moment transition and curvature learning in sequence. Learning the curvature without the first moment transition does not improve the average score: it changes from 40.75 ± 0.96 to 40.40 ± 0.40, a decrease of 0.35 that lies within the standard deviations of both variants. Adding the first moment transition raises it to 41.78 ± 0.35, a gain of 1.38 over CRAdam w/o MT that exceeds the standard deviations of both variants. Hence, the benefit of learnable curvature under Riemannian optimization depends on the first moment transition: relative to fixed curvature CRAdam, learning the curvature changes the average score by -0.35 without it and by $+1.03$ with it. The first moment transition is thus the component that allows learnable curvature to be exploited in CRAdam, rather than an independent addition to it.

H.3.2 Comparison with Bdeir-RAdam

We implement Bdeir-RAdam following the formulation of Bdeir et al. (2026) for comparison with our CRAdam. When the curvature is updated from c_t to c_{t+1}, each token embedding is mapped to the tangent space at the origin with the logarithmic map under c_t and mapped back to the manifold with the exponential map under c_{t+1}. The first moment is treated as a tangent vector and moved to the new point by two parallel transports in hyperbolic space: from the current point to the origin under c_t, and from the origin to the new point under c_{t+1}. Bdeir-RAdam and CRAdam are two complete optimizers for jointly learning hyperbolic parameters and curvature. They differ in the

curvature transition: Bdeir-RAdam transfers each embedding through the tangent space at the origin and moves the first moment by two parallel transports, whereas CRAdam preserves the spatial coordinates and rescales the radial component of the first moment in closed form. Since these choices are made jointly within each optimizer, we compare the two as complete methods.

With LBL, CRAdam outperforms Bdeir-RAdam over three random seeds, achieving 41.78 ± 0.35 compared with 40.78 ± 0.43 as provided in Table 6. Table 16 reports the detailed results summarized in Table 7, comparing the two optimizers under BMLR, hyperbolic geodesic logits (Variant IV), and LBL. All runs in this comparison use random seed 43; therefore, the LBL scores differ slightly from the three-seed averages in Table 6, although the ordering of the two optimizers is the same. CRAdam achieves a higher average score than Bdeir-RAdam under all three logits, with gains of 4.86, 0.92, and 2.07 points for BMLR, Variant IV, and LBL, respectively, showing that its advantage is not tied to a particular logit formulation. Moreover, LBL achieves the highest average score under both optimizers, and LBL with CRAdam gives the best overall result. CRAdam outperforms Bdeir-RAdam, both over three runs with LBL and under every evaluated logit formulation, while using a simpler curvature transition. We therefore adopt CRAdam for training the output layer of LBLM.

H.3.3 Riemannian Adam Efficiency

Implementation Details. CRAdam and Bdeir-RAdam are benchmarked on a single NVIDIA RTX4090 GPU using the same compilation setup. For both optimizers, the benchmark covers the complete update, including Riemannian gradient computation, moments update, bias correction, the manifold update, and the curvature transition. Each optimizer retains its own parameters and moments transition procedures. The measured runtime and peak additional memory therefore compare the complete implementations of the two optimizers, with the same compilation-based optimization applied to both.

For each method, we first perform 20 untimed warm-up iterations, followed by 200 measured iterations. Before measurement, the CUDA device is synchronized, unused cached memory is released, and the currently allocated memory is recorded as the baseline. Peak allocated CUDA memory is then tracked throughout the measured iterations. We report the increase from the pre-measurement baseline to the peak value in GiB. This quantity represents the peak additional CUDA memory during the benchmark rather than an average over iterations. Because the parameters, gradients, and initialized optimizer states are already included in the baseline, their existing storage is excluded from the reported value. The benchmark measures the complete optimizer update and does not separately attribute the observed difference to any individual geometric operation.

Memory and Runtime Efficiency. Figures 6c and 6d compare the complete update efficiency of our CRAdam and Bdeir-RAdam (Bdeir et al., 2026) as the hidden dimension increases from 1024 to 8192. As shown in Figure 6c, CRAdam requires less peak additional CUDA memory across all tested dimensions, and the gap increases with the hidden dimension. Since the parameters, gradients, and initialized optimizer states are already included in the baseline, this difference mainly reflects temporary tensors allocated during the optimizer update rather than persistent state storage. In Bdeir-RAdam, the update constructs multiple Lorentz points, tangent vectors, and momentum tensors of size approximately $|\mathcal{V}| \times (n+1)$. These include intermediate tensors for the manifold update, such as the direction, tangent step, updated Lorentz point, and momentum, as well as additional point and tangent representations introduced when mapping parameters and momentum states from the old curvature space to the new curvature space through logarithmic maps, exponential maps, and parallel transport. These intermediate tensors of size approximately $|\mathcal{V}| \times (n+1)$ account for the larger additional allocations during execution. In contrast, CRAdam performs the update primarily using the n-dimensional spatial coordinates and avoids these additional $(n+1)$-dimensional intermediate representations for transitions from the old curvature to the new curvature. Consequently, the difference in temporary memory becomes increasingly visible as n grows.

Figure 6d shows a similar trend in optimizer update time. CRAdam is faster than Bdeir-RAdam at every tested hidden dimension, and the gap widens as n increases: the update times of both optimizers grow approximately linearly with n, but that of Bdeir-RAdam grows more than twice as fast. This trend is consistent with the structure of the two updates. Neither update involves matrix multiplication; both consist of row wise inner products and elementwise linear combinations on tensors with one row per vocabulary token. Such operations are typically limited by memory

bandwidth rather than arithmetic on GPUs, so for a fixed vocabulary the update time grows linearly with n, at a rate determined by the number of passes over tensors of size approximately $|\mathcal{V}| \times n$ in each step. Under fixed curvature, the two optimizers perform comparable operations: both compute the Riemannian gradient, update the first moment, move each token embedding with the exponential map, and parallel transport the first moment to the updated point. They differ in the curvature transition. In Bdeir-RAdam, the transition consists of four geometric operations: each token embedding is mapped to the tangent space at the origin with the logarithmic map under c_t and back to the manifold with the exponential map under c_{t+1}, and the first moment is moved by two parallel transports, from the current point to the origin under c_t and from the origin to the new point under c_{t+1}. Each of these operations computes Lorentz inner products over $|\mathcal{V}| \times (n+1)$ tensors and writes a new tensor of the same size, so the transition alone involves more full size geometric operations than the exponential map and parallel transport of the fixed curvature update. The extra time coordinate in these $(n+1)$-dimensional representations enlarges each tensor only by a factor of $(n+1)/n$ and therefore cannot by itself explain the gap; the difference lies in the number of full size operations per step. In contrast, CRAdam keeps the spatial coordinates unchanged during the curvature transition and only rescales the radial component of the first moment (Eq. 101), without logarithmic maps or exponential maps. This rescaling has the same form as the Riemannian gradient in Eq. 62, so the transition adds a single lightweight operation to CRAdam instead of the four full size geometric operations of Bdeir-RAdam. This is consistent with the steeper growth of the update time of Bdeir-RAdam and with the widening gap between the two optimizers as embedding dimension n increases.