SUPPLEMENTARY FOR: PREDICATE HIERARCHIES IMPROVE FEW-SHOT STATE CLASSIFICATION

The appendix is organized as the following. In Appendix A, we include additional PHIER results, details, and discussion. In Appendix B, we describe implementation of baseline methods, including supervised models, pretrained large vision language models, and ablation variants. In Appendix C, we present preliminaries on hyperbolic geometry. In Appendix D, we detail prompts used to extract knowledge of predicates from LLMs. Finally, in Appendix E, we list all states in our datasets, and show examples from the BEHAVIOR Vision Suite Ge et al. (2024).

A PHIER RESULTS AND DETAILS

A.1 MODEL DETAILS

PHIER's image and text encoders are initialized with pretrained CLIP (Radford et al., 2021) and BERT (Devlin, 2018) weights, respectively. The hyperbolic linear layers are initialized following the approach of Shimizu et al. (2020), with the weights drawn from a normal distribution centered at zero with a standard deviation $(2nm)^{-\frac{1}{2}}$, where m and n are the input and output sizes of the layer, and the biases set to the zero vector. The linear layer in the small MLP is initialized by the standard Kaiming initialization. All of the parameters in PHIER are trainable and updated during training.

PHIER disentangles the conditioning of the image on the full state classification query into two distinct ones: one that identifies the relevant objects and another that focuses on key features for the given predicate. While we use MaskCLIP to identify the relevant entities, PHIER's contribution lies in the decomposition of the query into object and predicate components, enabling it to faithfully identify the relevant entities and extract features based on the predicate.

A.2 COMPARISON ON MANUALLY COLLECTED REAL-WORLD DATASET

We collect a small real-world dataset with 100 examples, consisting of 4 examples for each of the out-of-distribution BEHAVIOR states, to test our method's ability to perform zero-shot real-world transfer after training on simulated datasets alone. See Figure 4 for examples. In Table 5, we observe similar trends as in the BEHAVIOR Vision Suite evaluation in the main text, even with a simpler dataset. PHIER significantly outperforms prior supervised baselines. However, as expected, pre-trained models trained on large-scale real-world data outperform PHIER.

Figure 4: Examples from our manually collected real-world dataset.

A.3 ABLATION STUDY ON EXAMPLE COUNT

We study the effect of varying the number of examples used in the few-shot setting. We added new ablation experiments with 0, 1, 2, 3, 4, 5, and 10-shot generalization performance on both CALVIN and BEHAVIOR environments. The results in Figure 5 show that PHIER consistently outperforms prior works across all numbers of examples. Notably, in the CALVIN environment, PHIER's performance plateaus as the number of examples increases, indicating that the method requires only a few examples to adapt effectively to unseen scenarios.

Table 5: We present zero-shot generalization results of PHIER and prior works on a real-world test set, when trained only on the BEHAVIOR dataset. PHIER outperforms all prior supervised models.

	All	Unseen Combination	Novel Predicate
PHIER (Ours)	**0.62**	**0.64**	**0.60**
Re-Attention (Guo et al., 2020)	0.41	0.45	0.37
CoarseFine (Nguyen et al., 2022)	0.53	0.52	0.54
BUTD (Anderson et al., 2018)	0.44	0.42	0.46
RelViT (Ma et al., 2022)	0.55	0.59	0.51
CLIP (Shen et al., 2021)	0.55	0.65	0.45
FiLM (Perez et al., 2018)	0.49	0.51	0.47
GPT-4V (OpenAI, 2023)	0.72	0.74	0.70
BLIP-2 (Li et al., 2023b)	0.62	0.66	0.58
ViperGPT (Surís et al., 2023)	0.55	0.52	0.58

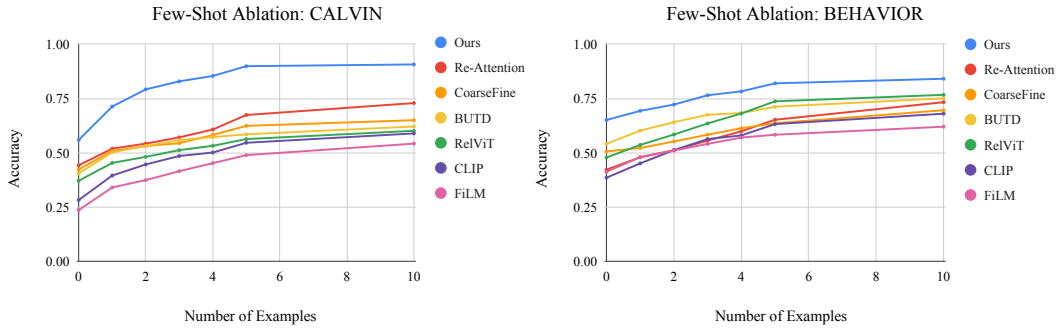

Figure 5: Ablations varying number of examples given in few-shot setting for CALVIN and BEHAVIOR environments.

A.4 Ablation Study with Removed Components

We add an ablation study that evaluates the impact of removing individual components of PHIER to evaluate their contributions. We compare PHIER with four variants, (1) without the object-centric encoder, (2) without the hyperbolic latent space, (3) without the norm regularization loss, and (4) without the predicate triplet loss. We report results in Table 6. We see that without our object-centric design, performance drops significantly in both ID and OOD settings, emphasizing the importance of object-centric encoders for improved representation and reasoning. In addition, we show that removing each of the self-supervised losses leads to much weaker generalization capability. Finally, we observe reduced generalization performance without PHIER 's hyperbolic latent space and hyperbolic norm regularization loss, demonstrating that the hyperbolic space facilitates better handling of hierarchical relationships. These results validate that each component contributes meaningfully to PHIER's performance, particularly in improving OOD generalization.

A.5 Few-shot Generalization to Novel Objects

We expand our CALVIN and BEHAVIOR experiments to evaluate accuracy on few-shot generalization on novel objects in Table 7. The queries with these novel objects are listed in Table 8. As in our experiments on unseen combinations and novel predicates, we observe that PHIER significantly outperforms prior baselines on unseen objects. Specifically, PHIER improves upon the top-performing prior work by 21.8 percent points on CALVIN and 13.5 percent point on BEHAVIOR. These results demonstrate that PHIER improves generalization to both novel objects and predicates, further highlighting the benefit of our object-centric encoder and inferred predicate hierarchy.

Table 6: Ablations of each component of PHIER and its effect on few-shot generalization.

	CALVIN			BEHAVIOR		
	ID ↑	OOD ↑	ID-OOD ↓	ID ↑	OOD ↑	ID-OOD ↓
PHIER (Ours)	0.945	0.899	0.046	0.859	0.820	0.039
- Object-centric encoder	0.786	0.704	0.082	0.703	0.659	0.044
- Predicate triplet loss	0.867	0.601	0.266	0.774	0.624	0.150
- Norm regularization loss	0.914	0.823	0.091	0.834	0.782	0.052
- Hyperbolic metric	0.903	0.784	0.119	0.803	0.761	0.042

Table 7: We present novel object generalization results of PHIER and prior works on CALVIN and BEHAVIOR environments.

	CALVIN	BEHAVIOR
PHIER (Ours)	**0.851**	**0.781**
Re-Attention	0.633	0.608
CoarseFine	0.562	0.632
BUTD	0.584	0.646
RelViT	0.497	0.642
CLIP	0.506	0.595
FiLM	0.411	0.521

Table 8: All states with novel objects (bolded) in the CALVIN and BEHAVIOR datasets.

Dataset	Predicate	Object 1	Object 2
CALVIN	OnTop	**red block**	table
	Stacked	**red block**	blue block
	Stacked	**red block**	pink block
	TurnedOn	**led**	–
BEHAVIOR	Inside	**box**	bottom cabinet
	Inside	**can**	bottom cabinet
	OnTop	**bottle**	breakfast table
	OnTop	**bottle**	**chair**
	OnTop	**box**	breakfast table
	OnTop	**bread**	breakfast table
	OnTop	**can**	**chair**
	Open	**refrigerator**	–

A.6 Visualizations on the Inferred Predicate Hierarchy

In Figure 6, we visualize the joint image-predicate space for BEHAVIOR on the Poincaré disk, highlighting the hierarchical semantic structure captured by PHIER's embeddings. By grouping the joint image-predicate embeddings by predicate, we uncover the inferred predicate hierarchy. For instance, we see that embeddings for `NextTo` are positioned closer to the origin compared to those for `OnLeft`, accurately reflecting their hierarchical relationship——`OnLeft` is a more specific case of `NextTo`. Furthermore, embeddings for `Touching` are nearest to the origin, consistent with its role as the most general predicate. For example, when one object is `Inside` or `OnTop` of another, they are inherently `Touching`. Similarly, objects that are `NextTo` or `OnLeft` are also frequently `Touching`. This visualization demonstrates that PHIER captures not only semantic structure but also nuanced hierarchical relationships between predicates.

We further analyze the embeddings for novel predicates after few-shot learning with only five examples. Notably, even with such limited data, PHIER successfully integrates these novel predicates

into the latent space and aligns them with their learned counterparts in semantically consistent regions (e.g., `OnRight` is near `OnLeft`). By aligning these predicates in similar regions, PHIER is able to leverage its existing knowledge of relevant features for learned predicates (e.g., `OnLeft`) to reason about novel predicates (e.g., `OnRight`). This alignment highlights that PHIER effectively encodes the relationships between pairwise predicates in the latent space, enabling generalization to novel predicates with minimal examples.

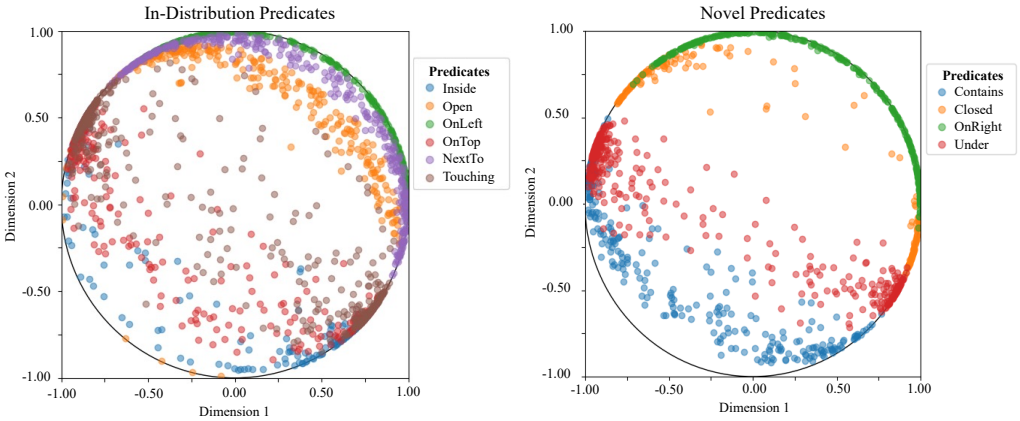

Figure 6: Visualizations of the joint image-predicate space for BEHAVIOR on the Poincaré disk, revealing that PHIER learns a meaningful predicate hierarchy. The novel predicate embeddings are visualized after few-shot learning with 5 examples.

A.7 In-Distribution Performance

Here, we discuss the in-distribution performance of PHIER in Table 1 of the main text. We note that in the in-distribution (ID) setting of CALVIN, PHIER outperforms all prior works except Re-Attention, with only a small margin of 1.4%. In the out-of-distribution (OOD) setting, which is our primary focus, PHIER outperforms Re-Attention by a significant 22.5%. Similarly, on ID BEHAVIOR, PHIER performs comparably to top-performing prior works, surpassing all except RelViT by 0.7%; however, in the OOD setting we focus on, PHIER outperforms RelViT by 8.3%. We highlight that PHIER performs comparably to top-performing prior works in the ID setting, while significantly improving the OOD performance. We focus on the few-shot generalization task and design our method to enforce bottlenecked representations (via a joint image-predicate space), while acknowledging that this might include tradeoffs on ID performance to avoid overfitting to the train distribution.

We also analyze specific cases where PHIER underperforms on ID examples. For instance, in CALVIN, we hypothesize that PHIER may struggle with tasks that the baselines may memorize due to their less constrained representations. We show an example in Figure 7, and note that for the ID query, `TurnedOn(lightbulb)`, Re-Attention correctly predicts `True`, while PHIER predicts `False`. However, for the out-of-distribution query, `TurnedOff(lightbulb)`, which is linguistically similar but semantically opposite, PHIER generalizes successfully while Re-Attention struggles to adapt. We conjecture that Re-Attention may predict that `TurnedOn(lightbulb)` is `True` based solely on the existence of the bulb at the location, instead of learning that the state of the lightbulb depends on its color (yellow is on and white is off). In contrast, we see that although PHIER's constrained representation may slightly limit learning capacity for ID settings, PHIER has the potential to conduct better compositional reasoning in OOD scenarios, where PHIER significantly outperforms baselines.

A.8 Object-centric Encoder Performance

We see empirically that PHIER's object-centric encoder performs well even on environments with significant distribution shifts, such as CALVIN. In Figure 8, we show an example of how the encoder localizes objects in CALVIN. To adapt to environments with even larger distribution shifts where the

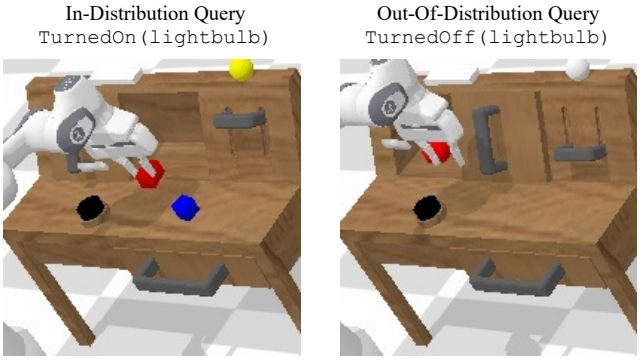

Figure 7: An example of the ID query, `TurnedOn(lightbulb)`, and OOD query with a novel predicate, `TurnedOff(lightbulb)`.

performance may decrease, we note that PHIER's object-centric encoder can be finetuned with more data as well.

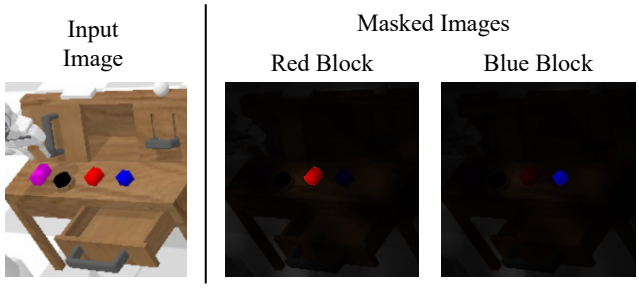

Figure 8: PHIER's object-centric encoder in the CALVIN environment.

B BASELINE DETAILS

For all of our baseline methods, we preprocess our input queries by converting the states into questions using the following templates:

- For unary states: "Is the {*object*} {predicate}"
- For binary states: "Is the {*object 1*} {predicate} the {*object 2*}"

B.1 SUPERVISED METHODS

We train all of the supervised baselines on the same training data as our method. Below, we describe each baseline and provide implementation details:

BUTD (Anderson et al., 2018). BUTD uses bottom-up attention to extract image features for important image regions and then top-down attention to focus on image regions based on the input query. We follow the original method, using Faster R-CNN pretrained on Visual Genome to extract bottom-up features for the top 36 image regions. For the text features, we embed the preprocessed text queries using 300-dimension word embeddings, initialized with pretrained GloVe vectors, and a GRU. The image and text features are then fed into model, based on the PyTorch implementation of the BUTD model for VQA (Yu et al., 2020)[*].

CLIP (Shen et al., 2021). We use pretrained CLIP vision and text encoders to extract features for the input image and query, respectively. These features are concatenated and passed through a small 2-layer network with a hidden layer of dimension 256 for state classification.

CoarseFine (Nguyen et al., 2022). Coarse to Fine learns to reason about scenes with complex semantic information by extracting image and text features at multiple levels of granularity. We follow the official implementation of the Coarse to Fine reasoning framework and use Faster R-CNN to extract image-level features and GRU with 300-dimensional GloVe embeddings to extract question-level features, which are then fed into the model[†].

FiLM (Perez et al., 2018). FiLM conditions an input image on text by applying learned transformations to the image features. We use a pretrained ViT-16 image encoder and BERT text encoder to extract image and query features. Then, a FiLM layer is applied to condition the image features on the query features, and the conditioned features are passed through a small 2-layer network with a hidden layer of dimension 256 for final prediction.

Re-Attention (Guo et al., 2020). Re-Attention introduces an attention mechanism to re-attend to objects in the images, based on the answer to the question. We follow the original implementation by using a Faster R-CNN model pretrained on the Visual Genome dataset to extract object-level image features, and 512-dimensional LSTM initialized with 300-dimensional GloVe embeddings to extract query features [‡].

RelViT (Ma et al., 2022). RelViT enhances the reasoning ability of vision transformers by introducing a concept-feature dictionary that enables efficient image feature retrieval during training. This supports a global task to promote relational reasoning and a local task to learn semantic object-centric correspondences. We use the official implementation, with Faster R-CNN to extract image region features, MCAN-Small as our VQA model, and the ImageNet1K-pretrained PVTv2b2 as our vision backbone [§].

SORNet (Yuan et al., 2022). SORNet extracts object-centric representations from input RGB images, conditioned on a set of object queries represented as images of the objects, to enable generalization to unseen objects on various spatial reasoning tasks. It performs state classification by training readout networks to predict spatial relations based on the object embeddings. For a fair comparison to our method and other baselines, we use MDETR (Kamath et al., 2021) to detect regions corresponding to object text, resize then to 32×32, and then use them as the input object images to train SORNet.

[*] https://github.com/MILVLG/bottom-up-attention.pytorch
[†] https://github.com/aioz-ai/CFR_VQA
[‡] https://github.com/gwy-nk/Re-Attention
[§] https://github.com/NVlabs/RelViT

We train readout networks for each training state in our dataset [¶]. Since SORNet requires training a separate network for each predicate, we only evaluate it on in-distribution states.

B.2 Pretrained Large Vision Language Models (VLM)

All of the pretrained large VLM baselines are evaluated inference-only.

BLIP-2 (Li et al., 2023b). We use BLIP-2 leveraging the OPT-2.7b language model and treat VQA as an open-ended answer generation problem. The input image is provided along with a query using the following format: "Question: {state query as a question} Answer:"

GPT-4V (OpenAI, 2023). We provide GPT-4V with the input image and a prompt based on the following template:

Prompt Template For GPT-4V Inference

```
Given an image of a scene, you will answer a question regarding the states
and relationships of objects in the scene.  The question is the following:

{state query as a question}

You need to carefully examine the image, thoughtfully consider the objects
in the scene, and analyze their states and relationships before answering
the question.

Provide your answer as True or False, and strictly follow this response
format:
Answer:  [insert your answer as True or False here]
Reasoning:  [insert your reasoning here]
```

Figure 9: Prompt template for GPT-4V experiments.

ViperGPT (Surís et al., 2023). We use the official ViperGPT implementation with Blip-2 Flan-T5 XXL as the pretrained model and GPT-4 for code generation. Our data is formatted according to the ViperGPT specifications, with the input image and query as a question.

B.3 Ablation Details

Here, we provide a clear breakdown of our ablation model architectures from Table 4 and explain how we add each component.

Supervised model. We start with a supervised baseline model, which uses an image encoder and text incoder initialized with CLIP and BERT weights, respectively.The embeddings from both encoders are concatenated and passed through a small MLP with three linear layers for classification, and the full model is trained with a binary cross-entropy loss based on the ground truth labels (True or False). We then progressively add each component of PHIER.

+ Object-centric encoder. First, we incorporate the object-centric encoder by replacing the image encoder, text encoder, and concatenation step with our proposed object-centric encoder, while retaining the MLP and loss.

+ Predicate triplet loss. Next, we introduce the predicate triplet loss by adding this term to the total loss function without changing the architecture.

+ Norm regularization loss. We further add the norm regularization loss to get the total loss function with all components, as described in Section 3.

+ Hyperbolic metric. Finally, we lift the scene representation to hyperbolic space using an exponential map and replace the first two linear layers in the MLP with two hyperbolic linear layers of the same size. We also use the Poincaré distance metric instead of the Euclidean metric in the self-supervised losses, yielding our final model (PHIER).

[¶] https://github.com/wentaoyuan/sornet

C HYPERBOLIC GEOMETRY PRELIMINARY

We briefly introduce the Poincaré ball model of hyperbolic space and hyperbolic neural networks. For a more detailed explanation, we refer the reader to Cannon et al. (1997) and Ganea et al. (2018).

As discussed in the main text, the Poincaré ball is a d-dimensional ball of radius 1, $\mathbb{P}^d = \{x \in \mathbb{R}^n : ||x|| < 1\}$, where $||\cdot||$ is the Euclidean norm. The ball is equipped with the metric tensor $g_{\mathbf{p}} = (\lambda_x)^2 g_{\mathbf{e}}$, where $\lambda_x = \frac{2}{1-||x||^2}$ is the conformal factor and g_e is the Euclidean metric tensor (i.e., the Euclidean dot product). This induces the Poincaré distance $\mathbf{d_p}$ between two points $x, y \in \mathbb{P}^d$ as follows:

$$\mathbf{d_p}(x, y) = \cosh^{-1}\left(1 + 2\frac{||x-y||^2}{(1-||x||^2)(1-||y||^2)}\right)$$

Möbius addition. On the Poincaré ball, Euclidean operations such as addition and multiplication have equivalents to ensure that all operations remain within the hyperbolic space and respect its geometry. Instead of using standard Euclidean addition, Möbius addition is used, which ensures that the sum of two points on the Poincaré ball still lies within the ball. The Möbius addition for any two points $x, y \in \mathbb{P}^d$ is defined as:

$$x \oplus y := \frac{(1 + 2\langle x, y\rangle + ||y||^2)x + (1 - ||x||)^2 y}{1 + 2\langle x, y\rangle + ||x||^2||y||^2}$$

Exponential and logarithmic maps. To perform operations in hyperbolic space, we use exponential and logarithmic maps to map Euclidean vectors to the hyperbolic space, and vice versa. For any point $z \in \mathbb{P}^d$, the closed form expression of the exponential and logarithmic maps centered around z are defined as:

$$\exp_z(y) = z \oplus \left(\tanh\left(\frac{\lambda_z||v||}{2}\right)\frac{v}{||v||}\right)$$

$$\log_z(y) = \frac{2}{\lambda_z}\tanh^{-1}(||-z \oplus y||)\frac{-z \oplus y}{||-z \oplus y||}$$

In practice, we use the maps centered at 0, $\exp_{\mathbf{0}}$ and $\log_{\mathbf{0}}$, to transition between Euclidean space and the Poincaré ball.

Hyperbolic neural networks. Ganea et al. (2018) proposes hyperbolic neural networks by defining hyperbolic equivalents of linear maps and bias translations. The hyperbolic linear map $M^{\otimes} : \mathbb{R}^n \to \mathbb{R}^m$ of any point $x \in \mathbb{P}^d$ on the Poincaré ball is defined as:

$$M^{\otimes}(x) = (1/\sqrt{c})\tanh\left(\frac{||Mx||}{||x||}\tanh^{-1}\left(\sqrt{c}||x||\right)\right)\frac{Mx}{||Mx||}$$

The translation of a point $x \in \mathbb{P}^d$ by a bias $b \in \mathbb{P}^d$ as:

$$x \oplus b = \exp_x\left(\frac{\lambda_{\mathbf{0}}}{\lambda_x}\log_{\mathbf{0}}(c)\right)$$

The hyperbolic linear layer is then defined as $M^{\otimes}(x) \oplus b$. To build a hyperbolic neural network, one simply has to map representations to the Poincaré ball using $\exp_{\mathbf{0}}$, apply hyperbolic linear layers, and then map back to Euclidean space using $\log_{\mathbf{0}}$.

Disk area of hyperbolic space.

We provide further details on why the exponential growth of the disc area in hyperbolic space provides a natural and efficient way to represent trees. Note that for a regular tree with a constant branching factor b, the number of nodes increases exponentially with the distance from the root, as $(b+1)b^{\updownarrow-1}$. We can embed trees in hyperbolic space, as they mirror this exponential growth. For instance, in a two-dimensional hyperbolic space with constant curvature $K = -1$, the circumference of a disc with radiuds r is $2\pi\sinh r$ while the area of a disc is $2\pi(\cosh r - 1)$. Since $\sinh r = \frac{1}{2}(e^r - e^{-r})$ and $coshr = \frac{1}{2}(e^r + e^{-r})$, both the circumference and area of the disc grow exponentially with the radius.

This exponential growth allows us to efficiently embed tree structures in hyperbolic space: nodes that are $\updownarrow$ levels from the root can be placed on the hyperbolic disc with a radius proportional to its level

$\updownarrow$, while nodes less than $\updownarrow$ levels within the sphere. Thus, we see how this property allows hyperbolic space to serve as a continuous representation of discrete trees.

Connection between hyperbolic space and hierarchical structure. We highlight several prominent prior works who have made theoretical connections between hyperbolic space and trees. Mathematical works such as Gromov (1987), Dyubina & Polterovich (2001), and Hamann (2018) prove that any finite tree can be embedded into a finite hyperbolic space with approximately preserved distances. A key property of hyperbolic space is its exponentially growing distance, and they show that this underlying property makes hyperbolic space well-suited to model hierarchical structures. Furthermore, works such as Sala et al. (2018) and Chami et al. (2020) propose concrete approaches to embed any tree in hyperbolic space with arbitrarily low distortion, establishing upper upper and lower bounds for distortion and further demonstrating the effectiveness of hyperbolic space for hierarchical modeling.

Notably, Nickel & Kiela (2017) were among the first to explore learning hierarchical representations in hyperbolic space. They found that for data with latent hierarchies, embeddings on the Poincaré ball outperform Euclidean embeddings significantly in terms of representation capacity and generalization ability. Since then, hyperbolic spaces have been increasingly explored for modeling hierarchies across various domains, including NLP (Ganea et al., 2018; Nickel & Kiela, 2018; Tifrea et al., 2018) and computer vision (Khrulkov et al., 2020; Ermolov et al., 2022), with substantial empirical evidence supporting its efficiency and suitability for modeling hierarchical structures in comparison to Euclidean space. We believe that these prior works provide strong theoretical justification and empirical support for the connection between hyperbolic space and hierarchical structure, which inspires our method.

Implementation. We implement our hyperbolic encoder using the Geoopt package (Kochurov et al., 2020), which provides functions and optimization methods for hyperbolic space [‖].

[‖] https://github.com/geoopt/geoopt

D LLM PROMPTS FOR SELF-SUPERVISED LOSSES

Here, we present the prompt templates used to extract explicit knowledge of predicates from LLMs. In Figure 10, we describe the prompt used to determine the assignment (anchor, positive predicate, and negative negative) for a given triplet of predicates, used in the predicate triplet loss. In Figure 11, we show the prompt used to determine the hierarchy among a predicate triplet based on specificity, for the norm regularization loss. We query the LLM once before training starts to retrieve the predicate triplet pairs and hierarchy, hence training is not affected by LLM queries.

Prompt Template For Predicate Triplet Assignment

```
You are given an anchor text query that describes a state of a scene.  Given
two other text queries describing the state of a scene, you will help
determine which of the two queries is more similar to the anchor query.

Consider the semantic meaning of the states and the specific aspects of
the scene they describe.  Additionally, think about how many objects and
what kinds of object properties and features you would need to verify if
evaluating these states against an image.

The anchor query is the following:  {anchor}

The other two queries are:
Query 1:  {query1}
Query 2:  {query2}

You must choose one of the queries as your answer.  Respond using the
following format:
Answer:  [Query 1 or Query 2]
```

Figure 10: Prompt template for inferring the predicate relations among a triplet with GPT-4.

Prompt Template For Triplet Hierarchy Ranking

```
You are an expert in scene understanding and state hierarchy determination.
Given three text descriptions each outlining a potential state of a
scene, your task is to establish a hierarchy among these descriptions by
identifying which one is the most general, which is the most specific, and
which lies in between.

Consider the following when determining the hierarchy:
- The variety and number of objects required by the state.
- The important features of the objects and/or relationships between the
objects.
- The level of detail provided about the scene.
- The semantic meaning of each description.

Your goal is to rank these descriptions in order of specificity, from least
specific (1) to most specific (3).

The three descriptions are:
1.  {anchor}
2.  {query1}
3.  {query2}

You must provide your ranking using the following format:
Least Specific:  [content of Description 1, 2, or 3]
Intermediate Specific:  [content of Description 1, 2, or 3]
Most Specific:  [content of Description 1, 2, or 3]
```

Figure 11: Prompt template for inferring the hierarchy among a triplet with GPT-4.

E Dataset Details

E.1 Dataset States

In Tables 9 and 10, we provide all of the states included in the CALVIN and BEHAVIOR datasets.

Table 9: All states included in the CALVIN dataset.

State Type	Predicate	Object 1	Object 2
ID	Lifted	blue block	–
ID	OnRight	slider	–
ID	Open	drawer	–
ID	TurnedOn	lightbulb	–
ID	Inside	blue block	drawer
ID	Inside	pink block	drawer
ID	OnTop	blue block	table
ID	Stacked	blue block	pink block
ID	Stacked	blue block	red block
OOD	Closed	drawer	–
OOD	Lifted	pink block	–
OOD	OnLeft	slider	–
OOD	TurnedOff	lightbulb	–
OOD	Inside	blue block	slider
OOD	OnTop	pink block	table
OOD	Stacked	pink block	blue block
OOD	Under	table	blue block

E.2 BEHAVIOR Vision Suite Visualizations

We include additional examples from BEHAVIOR Vision Suite Ge et al. (2024) in Figure 12.

Figure 12: Visualizations of state classification tasks from the real-world BEHAVIOR Vision Suite dataset.

Table 10: All states included in the BEHAVIOR dataset.

State Type	Predicate	Object 1	Object 2
ID	Open	bottom cabinet	–
ID	Open	drawer	–
ID	Open	microwave	–
ID	Open	oven	–
ID	Open	top cabinet	–
ID	Inside	apple	top cabinet
ID	Inside	club sandwich	microwave
ID	Inside	pizza	microwave
ID	Inside	plate	bottom cabinet
ID	Inside	plate	bottom cabinet no top
ID	NextTo	apple	coffee cup
ID	NextTo	coffee cup	cola bottle
ID	NextTo	croissant	cheesecake
ID	NextTo	pizza	microwave
ID	NextTo	plate	coffee cup
ID	OnLeft	apple	coffee cup
ID	OnLeft	coffee cup	cola bottle
ID	OnLeft	croissant	cheesecake
ID	OnLeft	pizza	microwave
ID	OnLeft	plate	coffee cup
ID	OnTop	apple	plate
ID	OnTop	cheesecake	plate
ID	OnTop	coffee cup	breakfast table
ID	OnTop	cola bottle	countertop
ID	OnTop	plate	breakfast table
ID	Touching	apple	plate
ID	Touching	cheesecake	plate
ID	Touching	coffee cup	breakfast table
ID	Touching	cola bottle	breakfast table
ID	Touching	croissant	plate
OOD	Closed	bottom cabinet	–
OOD	Closed	drawer	–
OOD	Closed	microwave	–
OOD	Closed	top cabinet	–
OOD	Contains	bottom cabinet	plate
OOD	Contains	drawer	plate
OOD	Contains	top cabinet	drawer
OOD	Inside	apple	microwave
OOD	Inside	coffee cup	top cabinet
OOD	Inside	plate	microwave
OOD	NextTo	apple	plate
OOD	NextTo	plate	microwave
OOD	OnTop	coffee cup	plate
OOD	OnTop	apple	breakfast table
OOD	OnTop	apple	microwave
OOD	OnLeft	apple	plate
OOD	OnLeft	coffee cup	apple
OOD	OnRight	apple	coffee cup
OOD	OnRight	coffee cup	cola bottle
OOD	OnRight	plate	coffee cup
OOD	Touching	apple	breakfast table
OOD	Touching	coffee cup	plate
OOD	Under	breakfast table	coffee cup
OOD	Under	breakfast table	plate
OOD	Under	plate	apple