CS 584– Natural Language Processing

Prof. Ping Wang

STEVENS
INSTITUTE OF TECHNOLOGY
1870

Reasoning as Energy Minimization: An Energy Based Approach to Question Answering

Team 3

- **Milan Chandiramani,**
- **Syed Yasir Alam**

Dec 01, 2025

Team Members

Milan G Chandiramani

CWID: 20032010

Syed Yasir Alam
CWID: 20031820

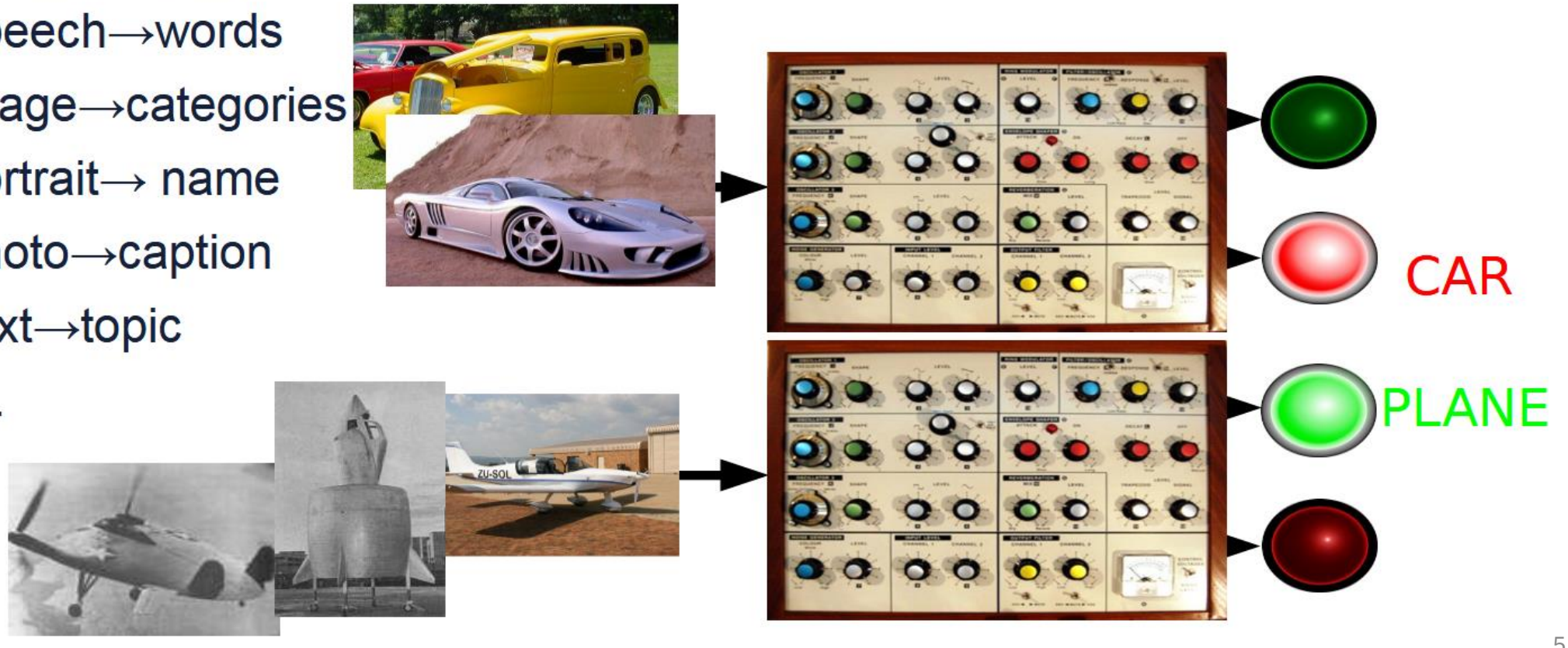
Table Of Contents

Introduction

Probability distributions are complex, probabality prediction can average to hallucination for predicting multiple outputs, how about we consider energy?

Motivation: Supervised Learning works but requires many labeled samples

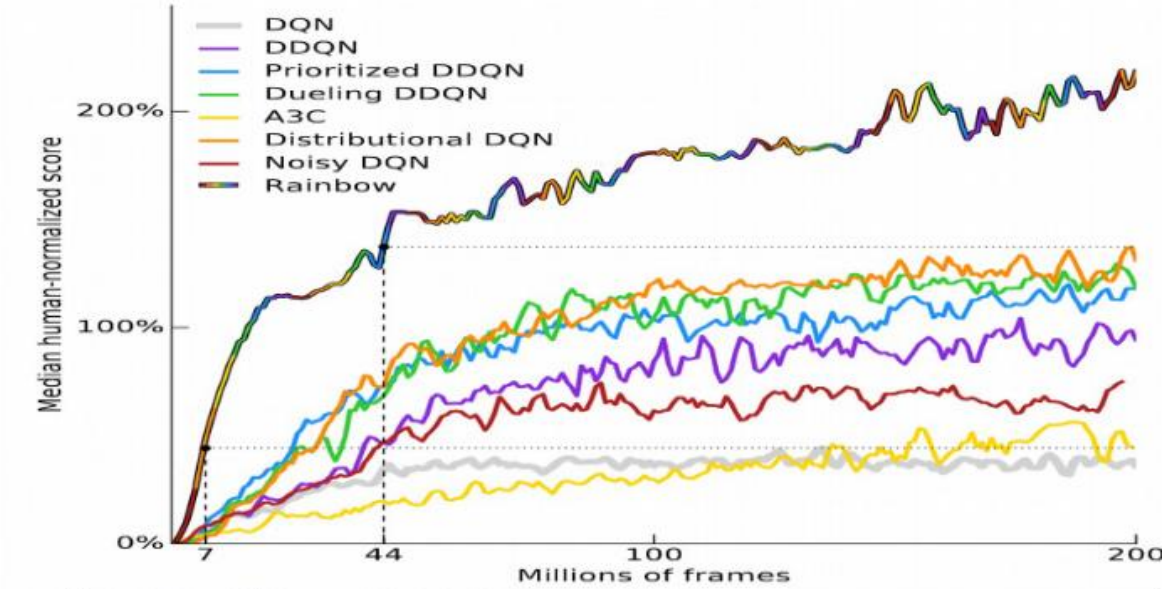
- **Training a machine by showing examples instead of programming it**
- **When the output is wrong, tweak the parameters of the machine**
- **Works well for:**
  - Speech→words
  - Image→categories
  - Portrait→ name
  - Photo→caption
  - Text→topic
  - ….

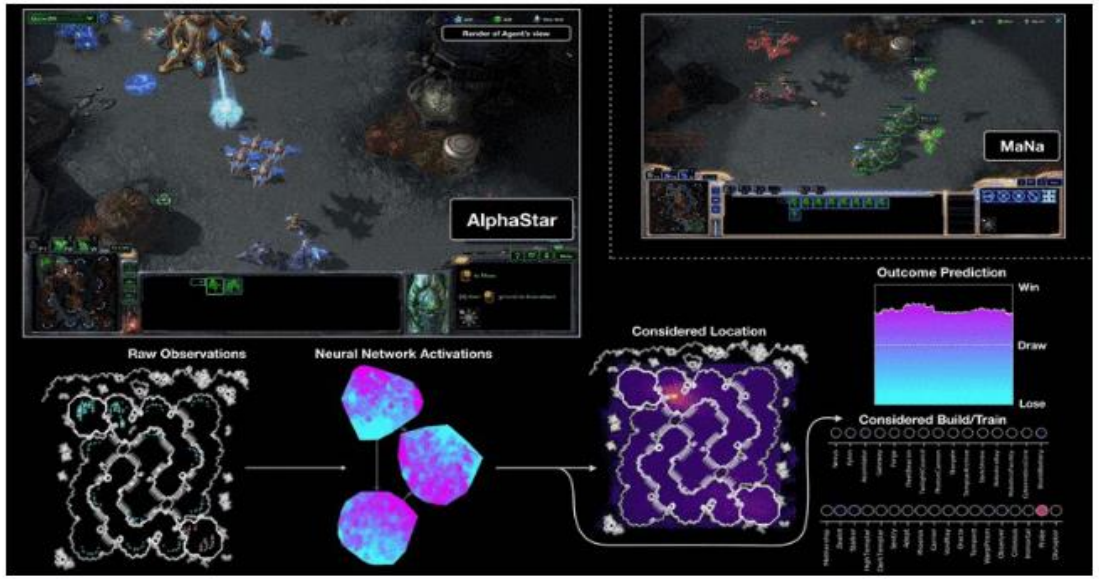


- Referenced from Yann LeCun: "Energy-Based Self-Supervised Learning" at IPAM

Reinforcement Learning: can work great for simulations

- **57 Atari games: takes 83 hours equivalent real-time (18 million frames) to reach a performance that humans reach in 15 minutes of play.**
  - [Hessel ArXiv:1710.02298]
- **Elf OpenGo v2: 20 million self-play games. (2000 GPU for 14 days)**
  - [Tian arXiv:1902.04522]
- **StarCraft: AlphaStar 200 years of equivalent real-time play**
  - [Vinyals blog post 2019]
- **OpenAI single-handed Rubik's cube**
  - 10,000 years of simulation





* Referenced from Yann LeCun: "Energy-Based Self-Supervised Learning" at IPAM

Reinforcement Learning Drawback: Requires Lot of Trials

- Pure RL requires too many trials to learn anything
  - it's OK in a game
  - it's not OK in the real world
- RL works in simple virtual world that you can run faster than real-time on many machines in parallel.
- Anything you do in the real world can kill you
- You can't run the real world faster than real time

How do humans and animals learn so quickly?

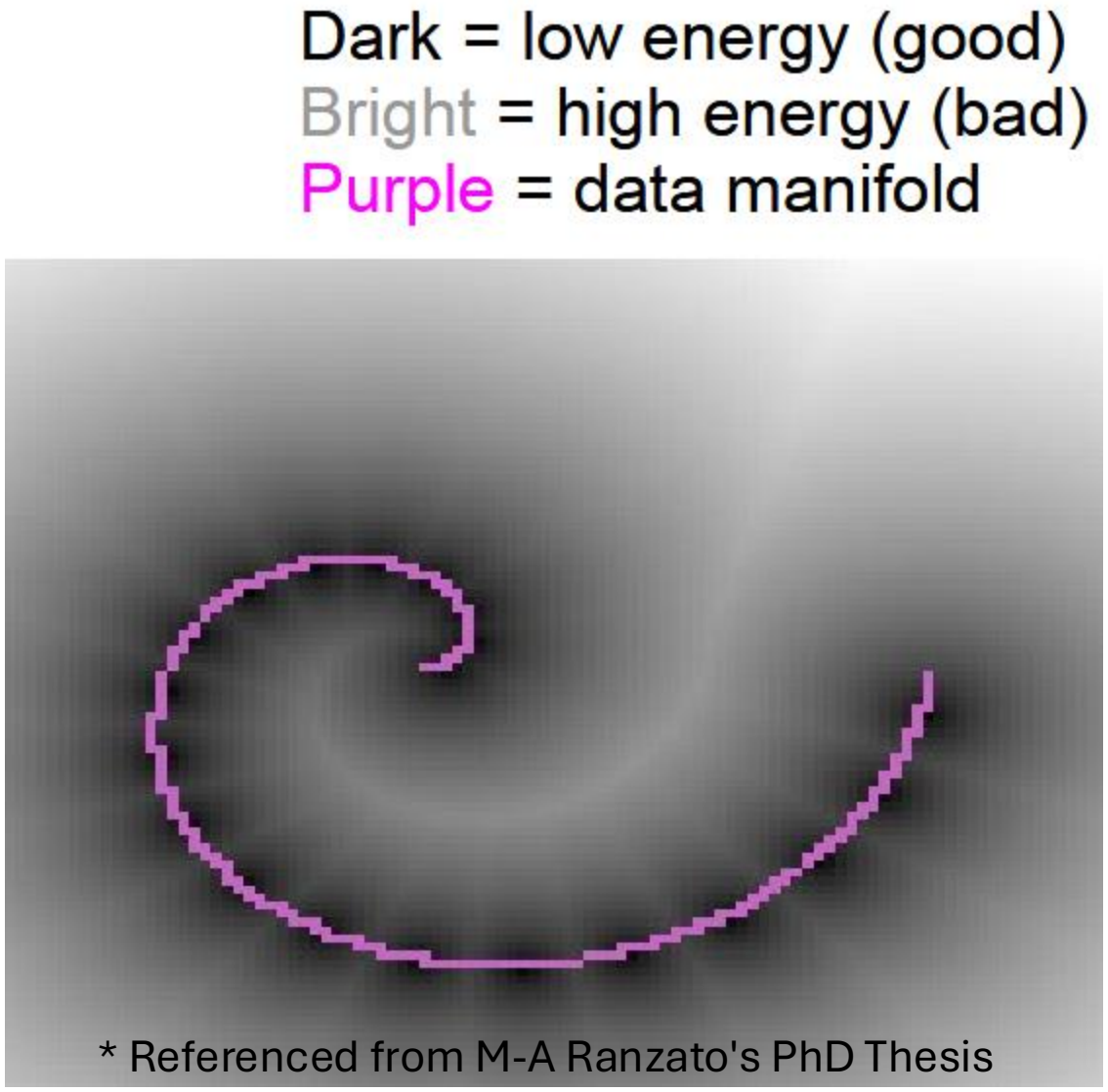
Not Supervised

Not Reinforced

Largely by observation, with remarkably little interaction.

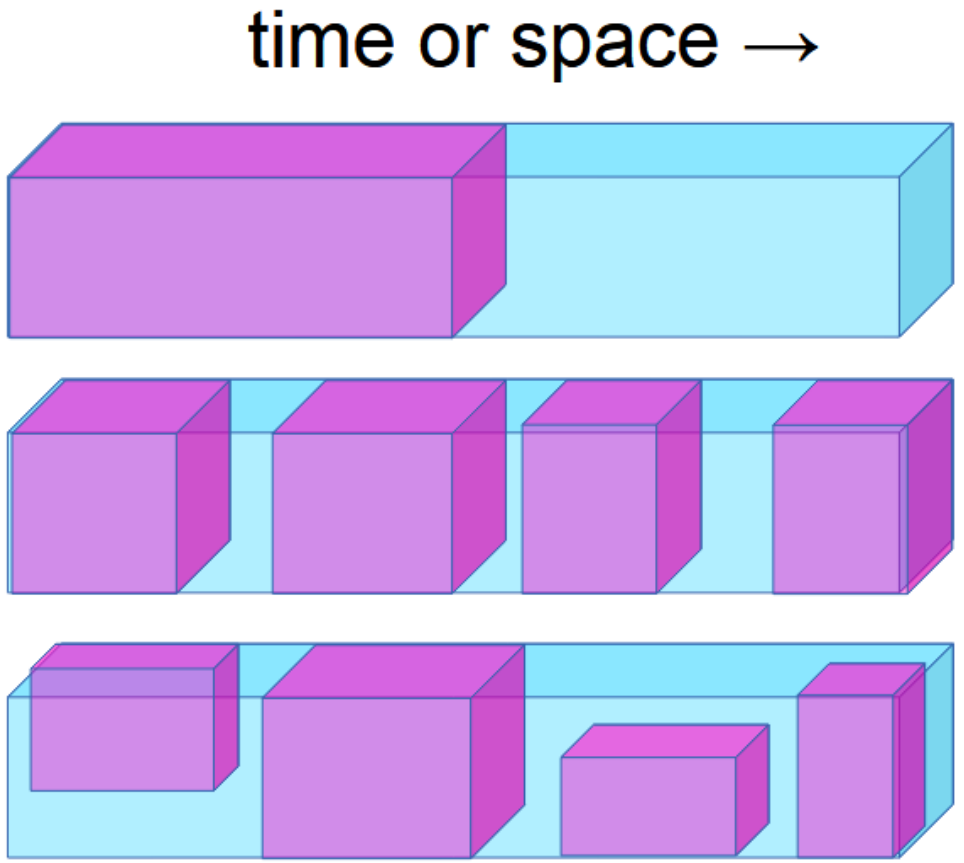
Problem Statement & Objective

- Current question-answering systems mix reasoning and text generation, often failing at deep thinking.
- We aim to **separate reasoning from fluency**:
  - **Reasoning:** Modeled as an optimization process in latent space using Energy-Based Models (EBMs).
  - **Fluency:** Handled by a language decoder for natural answers.
- Our approach:
  - Use **energy minimization** to refine answers iteratively.
  - Apply advanced training techniques (gradient truncation, memory buffers, use High Variational Auto-Encoders (HVAE))



Self-Supervised Learning = Filling in the bl_nks

- Predict any part of the input from any other part.
- Predict the future from the past.
- Predict the masked from the visible.
- Predict the any occluded part from all available parts.



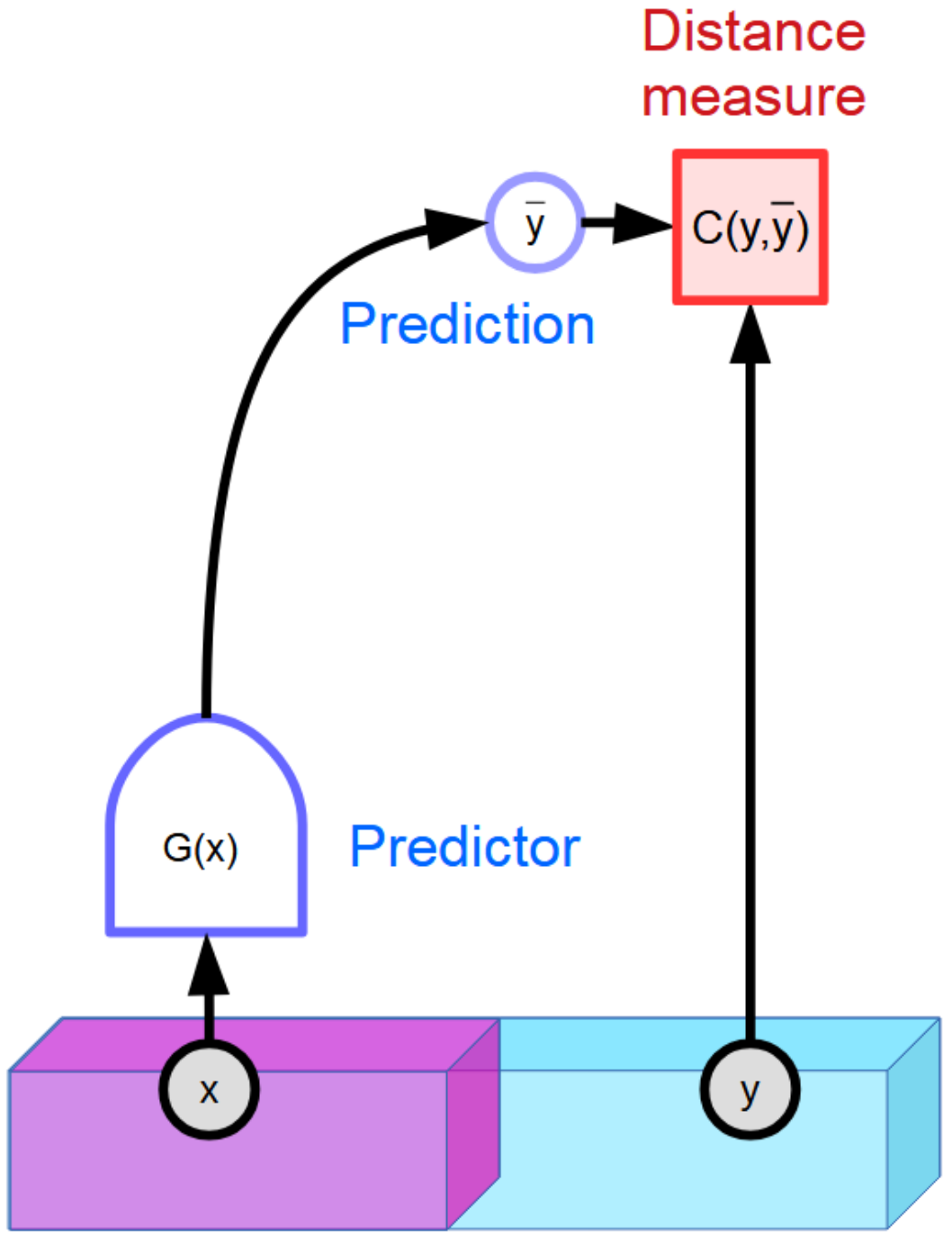
- Pretend there is a part of the input you don't know and predict that.

* Referenced from Yann LeCun: "Energy-Based Self-Supervised Learning" at IPAM

Energy Based Models

- Learning to deal with uncertainty while eschewing probabilities
- There are many plausible words that complete a text.
- There are infinitely many plausible frames to complete a video.
- Deterministic predictors don’t work!
- How to deal with uncertainty in the prediction?

$$E(x, y) = C(y, G(x))$$



* Referenced from Yann LeCun: "Energy-Based SSL" at IPAM

Question Answering Challenges

- While this field has seen explosive growth with the advent of the Transformer architecture. Current question-answering systems mix reasoning and text generation, often failing at deep thinking.
- **Ambiguity and Context:** Natural language is highly ambiguous (e.g., the word "bank"). Systems must accurately understand the meaning based on the complete context and conversational history, which is difficult.
- **Reasoning and Inference:** Answering questions that require logical deduction or multi-step reasoning, rather than simple information extraction.
- **Data Quality and Bias:** Models are only as good as their training data. Biased, low-quality, or domain-specific data can lead to inaccurate, unfair, or unreliable results.
- **Factuality and Hallucination:** Generative models sometimes produce fluent, but entirely incorrect or fabricated information ("hallucinations"). This is a primary focus of research.
- **Multilingualism:** Ensuring high-quality QA performance for all the world's languages, especially those with limited training data or complex grammar.

Article: Super Bowl 50
Paragraph: "*Peyton Manning became the first quarterback ever to lead two different teams to multiple Super Bowls. He is also the oldest quarterback ever to play in a Super Bowl at age 39. The past record was held by John Elway, who led the Broncos to victory in Super Bowl XXXIII at age 38 and is currently Denver's Executive Vice President of Football Operations and General Manager. Quarterback Jeff Dean had jersey number 37 in Champ Bowl XXXIV.*"
Question: "*What is the name of the quarterback who was 38 in Super Bowl XXXIII?*"
Original Prediction: John Elway
Prediction under adversary: Jeff Dean

LLMs are great, but ...

- The LLMs ability to generate fluent and contextually-aware text is unparalleled.
- However, this autoregressive, generative-first approach suffers from two fundamental weaknesses.
  - LLMs struggle with **complex reasoning** and exhibit limited generalization to unseen domains or problems more complex than those in their training data.
  - Their computational cost for inference is fixed; a model like GPT-4 expends the same amount of compute to the answer "what is the capital of France?" as it does to solve a multi-step logical puzzle. This fixed-cost inference prevents them from "thinking harder" about more difficult problems.
- This limitation highlights a critical distinction between solution generators and solution verifiers. For complex reasoning problems, the computational resources required for a generator to find a correct solution can scale exponentially with problem difficulty. In contrast, the resources required for a verifier to simply check if a given solution is correct often scale polynomially.

Contributions

- As part of this project, we are exploring use of energy-based models and their adoption for a more robust and generalizable reasoning system can be built by adopting an EBM-centric, verification-first approach.
- Energy-Based Models (EBMs) learn a scalar energy function that assigns low energy to correct or plausible solutions and high energy to incorrect ones. Recent work has demonstrated that this iterative, energy-based paradigm provides a powerful framework for generalization. Studies such as Learning Iterative Reasoning through Energy Minimization, Learning Iterative Reasoning through Energy Diffusion, and Generalizable Reasoning through Compositional Minimization have shown that EBMs can effectively generalize to new, more complex problems in a way that end-to-end generative models cannot.
- Despite this promise, the adoption of deep EBMs has been historically hindered by their notoriously unstable and computationally expensive training dynamics. To build a system that is both powerful and practical, we must try to overcome these hurdles.
- We utilize HVAE combined with EBM to establish a robust, reasoning-based Q/A system

Project Challenges

```
OutOfMemoryError: CUDA out of memory. Tried to allocate 5.50 GiB. GPU 0 has a total capacity of 15.89 GiB of which 3.87 GiB is free. Process 5981 has 12.01 GiB memo
ry in use. Of the allocated memory 9.65 GiB is allocated by PyTorch, and 2.06 GiB is reserved by PyTorch but unallocated. If reserved but unallocated memory is larg
e try setting PYTORCH_CUDA_ALLOC_CONF=expandable_segments:True to avoid fragmentation.  See documentation for Memory Management  (https://pytorch.org/docs/stable/no
tes/cuda.html#environment-variables)
```

- Compute Constraints.
- Large-Scale Multi-Component Design leading to multiple unexpected failures.
- Dataset Domain Shift.
- Limited Engineering Guidance for EBMs.

Datasets Considered

Datasets we have considered so far

SQuAD: Stanford Question Answer Dataset

- **Purpose:** Designed to test machine reading comprehension
- **Content:** Over 100,000 ques tions based on Wikipedia articles
- Models must answer questions by extracting text spans from the giwen passage
- Versions:
  - SQuAD 1.1: Questions with answers found in the passage
  - SQuAD 2.0: Includes unanswerabble questions to test reasoning
- **Impact:** A standard benchmark for NLP research and QA systems.

Hugging Face H4 stack-exchange-preferences

- Stack Overflow Data Dump
- More than 10 million rows

- We have focused on Physics Stack Exchange Dataset
- ~36,000 training samples (after QA pair matching)

PHYSICS

qid int64	question string · *lengths*	answers list	date string · *lengths*	metadata list
1↔842k 8.8%	18↔4.84k 98.6%		10 100%	
1	<p>I have been wanting to learn about 3D printing a lon g time so I really want this site to succeed but I hav…	[{ "answer_id": 14, "author": "Eric Johnson", "author_id": 43, "author_pr… {…}	2016/01/12	["https://3dprinting.meta.stackexchange.com/ questions/1",…

Model : Energy based (Transformer Backed) Model

How we went about energy based VAEs

EBM Stage 1: Data Processing

- We use SQuAD dataset and generate SQuAD tensors (stored in .pt) containing fields px, hx, ax_gold.
- QALatentDataset wraps the loaded dictionary-style tensor data.
  __len__ uses len(self.data["input_ids"]) and __getitem__ returns a dict with "px", "hx", "ax_gold".
  - px : tokenized input or some additional precomputed features,
  - hx : latent/context or question representation (used as conditioning input),
  - ax_gold : gold (true) answer embedding/vector.

EBM Stage 2: Model Architecture and Parameters

- **Input:** pre-generated SQuAD tensors (loaded from `.pt`) containing fields `px`, `hx`, `ax_gold`.
- **Model Architecture:** Energy-Based Model (EBM Model) is built from:
- `soft_token_generator` (MLP → GELU → LayerNorm) → produces *soft tokens* from input vectors that maps answer vectors into num_tokens soft token embeddings via a small MLP (soft_token_generator). Those soft tokens are concatenated (question tokens + answer tokens) then passed through a Transformer encoder.
- `TransformerBackbone` (Linear input projection → PositionalEncoding → TransformerEncoder) : After transformer encoding, the model pools or reads an appropriate representation (Transformer o/p) and passes those embeddings through energy_head to get a scalar energy. Lower energy is meant to indicate a better answer.
- `energy_head` (MLP → tanh → Linear) → outputs scalar energy

EBM Stage 3: Inference and Loss

- Markov Chain Monte Carlo (MCMC) style *refinement* of candidate answer vectors by computing ∇(energy) w.r.t. the answer vector and updating the vector (gradient steps + noise).
- MCMC refinement trains corrupt candidate, compute energy, compute gradient of energy w.r.t. candidate vector. It also normalizes gradient,
  step refined vector : (refined_vec = refined_vec - alpha * grad_normalized); so moving candidate in energy-reducing direction or as specified and collects refined_vec_history, predicted_energies
- **Losses:** reconstruction MSE between refined vector and true answer vector; contrastive loss computed by comparing energy of true vs refined answers (cross-entropy on stacked energies).
  `Loss = recon_coeff * MSE + contrast_coeff * CE`.

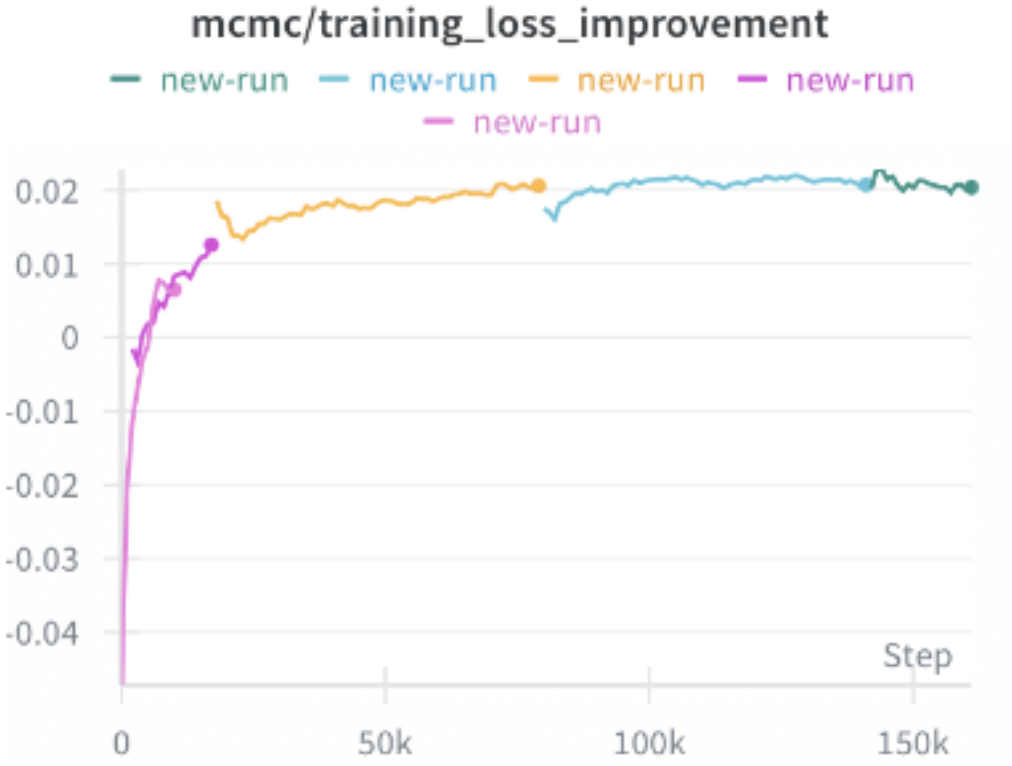
- After refinement, we compute forward reconstruction losses (MSE between final refined vector and gold answer vector), builds loss history across steps; also computes contrastive loss by comparing energy of real vs fake pairs.
- We calculate contrastive loss to get energy for the real (gold) answer and for the final refined candidate, stacks them and applies Cross Entropy on -energy_stack with targets pointing to the real pair so lower energy for real should give higher score. This is effectively a contrastive energy ranking loss.

EBM Stage 3: Optimization

- **Optimizers & scheduler:** We use AdamW with separate lr groups (soft_token_generator, backbone, energy_head), and `LinearWarmupCosineAnnealingLR` schedule.
- **Logging / checkpointing:** Weights & Biases (wandb) used for run tracking; model checkpoints saved to disk. Optional replay buffer used for candidate vectors.
- **Metrics reported:** average MSE (initial vs refined) on test batch, various wandb tables for energy / grad norms / reconstruction progression.

EBM Challenges

- Our model got stuck because we were trying to learn smooth energy patterns using variables that were not smooth.
- Imagine trying to draw a smooth curve using only Lego blocks the shapes are too rigid, so the model cannot learn the gentle slopes it needs.
- Because of this mismatch, the Energy-Based Model could not learn a meaningful energy landscape."
- EBMs require a *continuous* latent space so that the energy function is differentiable with respect to the input.
  However, our latent answer vectors were *discrete* or piecewise-constant.
  This means:
- ∂E/∂x is either 0 or undefined,
- MCMC refinement cannot meaningfully follow gradients,
- The model never learns a smooth or valid energy surface.
  Therefore, gradient-based refinement collapses and the EBM cannot learn useful structure."



(a) MSE Improvement (Training Steps)

Model : Hierarchical Variational Autoencoder (HVAE) Architecture

How we went about energy based VAEs

May 06, 2025

HVAE and EBT Infusion

- **Goal :** Given a QnA pair, we try to minimize energy and verify if the QnA pair is valid.
- **Steps:**

1. We train the HVAE encoder-decoder model to learn the encoding-decoding latent for answers.

2. We train the EBT that will take the question and generate answer from random noise and the vector of a given distribution (μ,σ^2) is decoded through HVAE decoder and the final answer is generated. Role of EBT is to learn the energy space latent and identify a high-dimensional vector sample from the unnormalized distribution such that it fulfills the constraints of the question (context).

- Treat each post/answer/abstract as a hierarchical text object made of sentences, where each sentence is a sequence of words/tokens.
- We take these QnA sentences and use HVAE to encode the discrete sentences to continuous latent space using local and global latents.

HVAE Stage 1: Dataset Processing

- Goal : Treat each post/answer/abstract as a hierarchical text object made of sentences, where each sentence is a sequence of words/tokens.
- High-level steps in preprocessing:
  - **Data Cleaning :** Question with multiple Answers are provided. We identify the best answer, clean the question answer pair.
  - **Sentence segmentation :** Each document is split into sentences using an NLTK-style sentence tokenizer. Sentences are joined with a special <EOS> token so later steps can split into sentences easily. We do this as HVAE is hierarchical; it needs sentence boundaries explicitly so the model can build word-level encodings inside sentence-level encodings.
  - **Filtering by length :** We filter documents to keep abstracts whose number of sentences fall within configured bounds (e.g. between 5 and 11 sentences in an example cell). This keeps the batch shapes stable and avoids very long examples.
  - **Typical parameters used:** max_sentences = 11, max_words = 30 (see hyperParams).
  - **Tokenization** : Uses a GPT-2 tokenizer (or a GPT-2-like tokenizer) and extends its vocabulary (we update vocab_size, pad_index, mask_token_id after adapting the tokenizer). Documents are converted into the hierarchical token structure: a list (sentences) of lists (word/token ids), padded/truncated to (max_sentences, max_words) per example. The tokenizer PAD / EOS / MASK token ids are stored in hyperParams for use across the model/loss.
  - Dataset class and DataLoader : HVAEDataset yields batched examples already arranged as sentence lists and masks. A PyTorch DataLoader is created with chosen batch size.

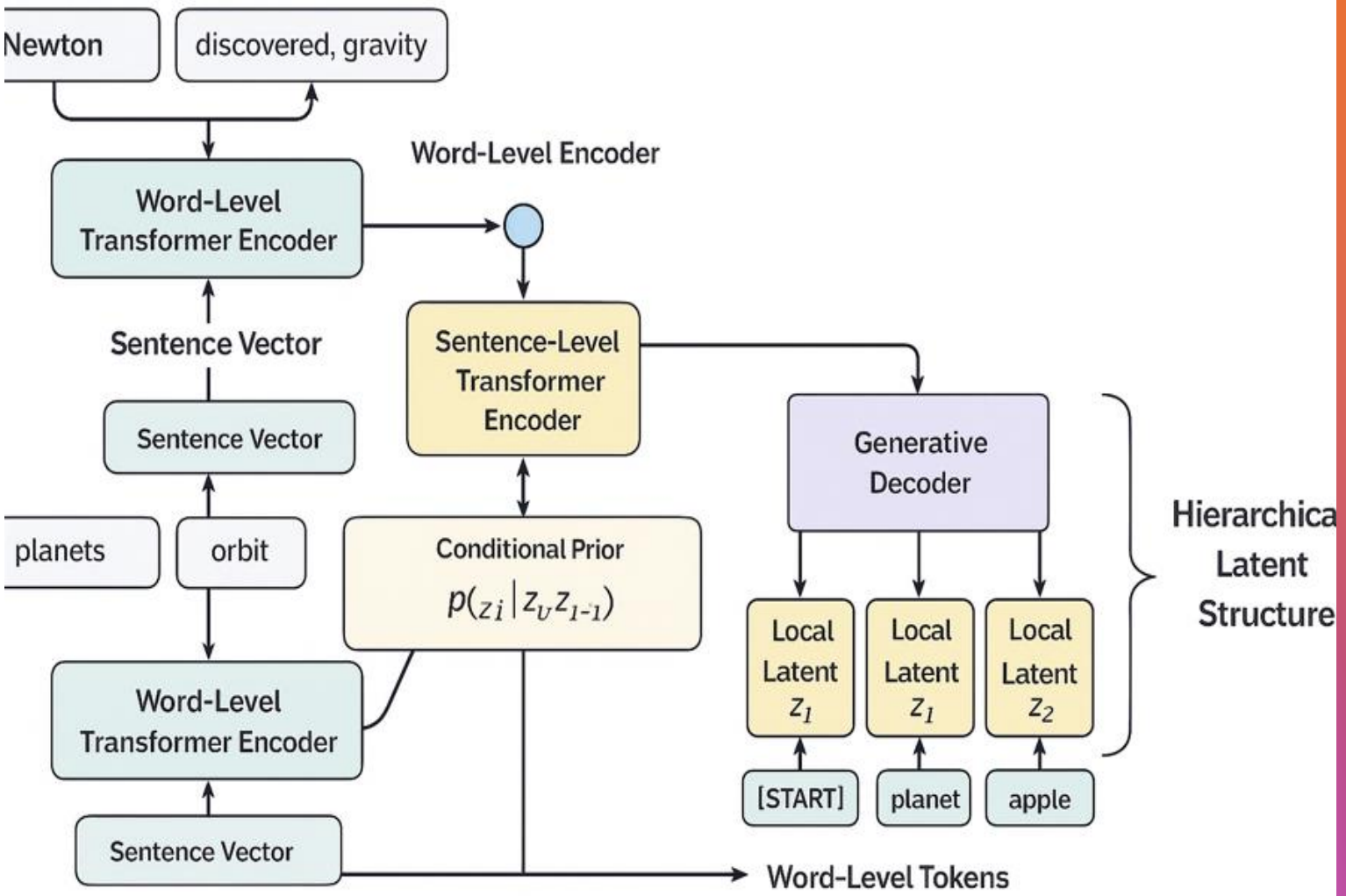
	question	answer	question_clean	answer_clean	num_sentences
1	<p>How would you explain string theory to non-...	<p>I've noticed that none of these answers act...	How would you explain string theory to non-phy...	I've noticed that none of these answers actual...	6

HVAE EBT Stage 2: Model Architecture and Parameters

- 2 Hierarchy levels:
  - **Word-level** (within a sentence): transformer encoder produces per-sentence encodings.
  - **Sentence-level** (across sentences): a separate transformer encodes sentence representations into global context.
- Latent variables:
  - **Global latent** (z_t) : (a single high-level vector for the whole document / abstract).
  - **Local latents** (z_i) : (one per sentence) that are conditionally dependent on the global latent and the previous local latent(s) in the chain i.e., a chain prior to capture temporal/ordered dependencies across sentences.

HVAE Stage 2: Model Architecture and Parameters

- Main Model Components:
- **Word-Level Transformer Encoder:**
  - Encode token into sentences.
- **Sentence-Level Transformer Encoder:**
  - Summarize the document
- **Inference network (encoder → posterior)**
  - MLPs take sentence representations & o/p: (μ, σ^2) for each local z_i & global z_t
  - Compress Input Seq into (μ, log (σ2)) for the Global and Local Gaussian distributions
  - Reparameterization trick reparameterize(μ,σ2) is used to sample z during training: $z=\mu+\sigma\odot\epsilon$.
- **Generative network (decoder)**
  - Uses sampled latent variables (zg and zi) & reconstructs original o/p seq (target answer).



HVAE Stage 2: Model Architecture and Parameters

- Key hyperparameters:
- `'gpt2_model_name': 'gpt2'` : base tokenizer/embedding expectations (d_model = 768).
- `'d_model': 768` : transformer hidden size.
- `'latent_dim': 128` : dimension of each latent vector (global and local). (Typical range: 32–128)
- `'max_sentences': 11`, `'max_words': 30` : hierarchical shape limits.
- `'encoder_layers': 2`, `'encoder_heads': 8`, `'encoder_dropout': 0.1'`
- `'gru_layers': 1'` : for high-level sentence GRU (if used).
- `'pad_index'`, `'mask_token_id'`, `'word_dropout_rate': 0.3'`
- `'batch_size_for_creation' : 64`
- `'target_batch_size' : 64`

HVAE Stage 3: Loss Functions & Train Configuration

- **Loss=Reconstruction Loss+β·KL Divergence Loss**
- **Reconstruction loss (Negative log-likelihood)**
  - Implemented with CrossEntropyLoss(ignore_index=pad_idx, reduction='none') across tokens.
  - Measures how well the decoder reconstructs each token in the sentence(s). Padding tokens are ignored.
  - Often averaged to a per-token loss for diagnostics.
- **Global KL** (between posterior $q(z_t \mid x)$ and a prior $p(z_t)$)
- This term pushes the global latent distribution to stay close to prior; controls capacity of global latent.
- **Local KL** (sum across sentence latents KL(q(z_i|x) || p(z_i | z_t, z_{i-1})))
- Each sentence latent has its own KL versus its *conditional* prior.
- As prior depends on z_t and previous z, this KL enforces the local latents to be consistent with the document-level/global level plan and the sentence ordering.

HVAE Stage 3: Loss Functions & Train Configuration

- **KL annealing (β schedules)** : KL weights `global_kl_beta` and `local_kl_beta` are gradually increased from 0 → 1 to avoid posterior collapse and to let reconstruction stabilize early in training.
- **Free bits (KL floor)** : We apply a *free-bits* threshold. local KL values below a threshold are floored (i.e., not penalized) to encourage the latent to have a minimum information capacity. This is implemented as `local_free_bits` and applied before summation.
- **Masking of losses** : Sentence and token-level masks ensure that KL & reconstruction are only computed on real tokens/sentences (not padding).
- **Word dropout / masking in inputs** : Randomly mask words in the encoder input at rate `word_dropout_rate` to encourage the model to rely on latent structure rather than exact words.

- **Optimizer and scheduler (training configs)**
- Optimizer: `AdamW` (typical decoupled weight decay optimizer for transformers).
- Learning rate schedule: a combination of warmup and cosine/linear schedules; we try to combine `SequentialLR`, `LinearLR`, and `CosineAnnealingLR` to implement warmup followed by annealing.

HVAE Stage 4: Inference & generation (decoder behavior)

- **How generation is implemented:**
- Generation loops over sentences: for each sentence, the model samples/decodes tokens autoregressively until `<EOS>` or `max_words` reached.
- For each sentence's generation, decoder generates plan vector using the sampled local latent z_i and the global latent z_t to condition the decoder. After finishing a sentence, z_i may be used to compute $z_{\{i+1\}}$ prior (for conditional prior consistency).
- Output is a nested list of token ids/plan vectors `[[sent1_tokens], [sent2_tokens], ...]` which are fed to GPT-2 for each sentence, detokenized back to text and generates the valid sentence.

- **Important to understand:**
- The decoder is not a simple flat unconditional generator — it conditions each sentence generation on explicit latent variables which encode both global style/intent and sentence-specific content.

HVAE Metrics :

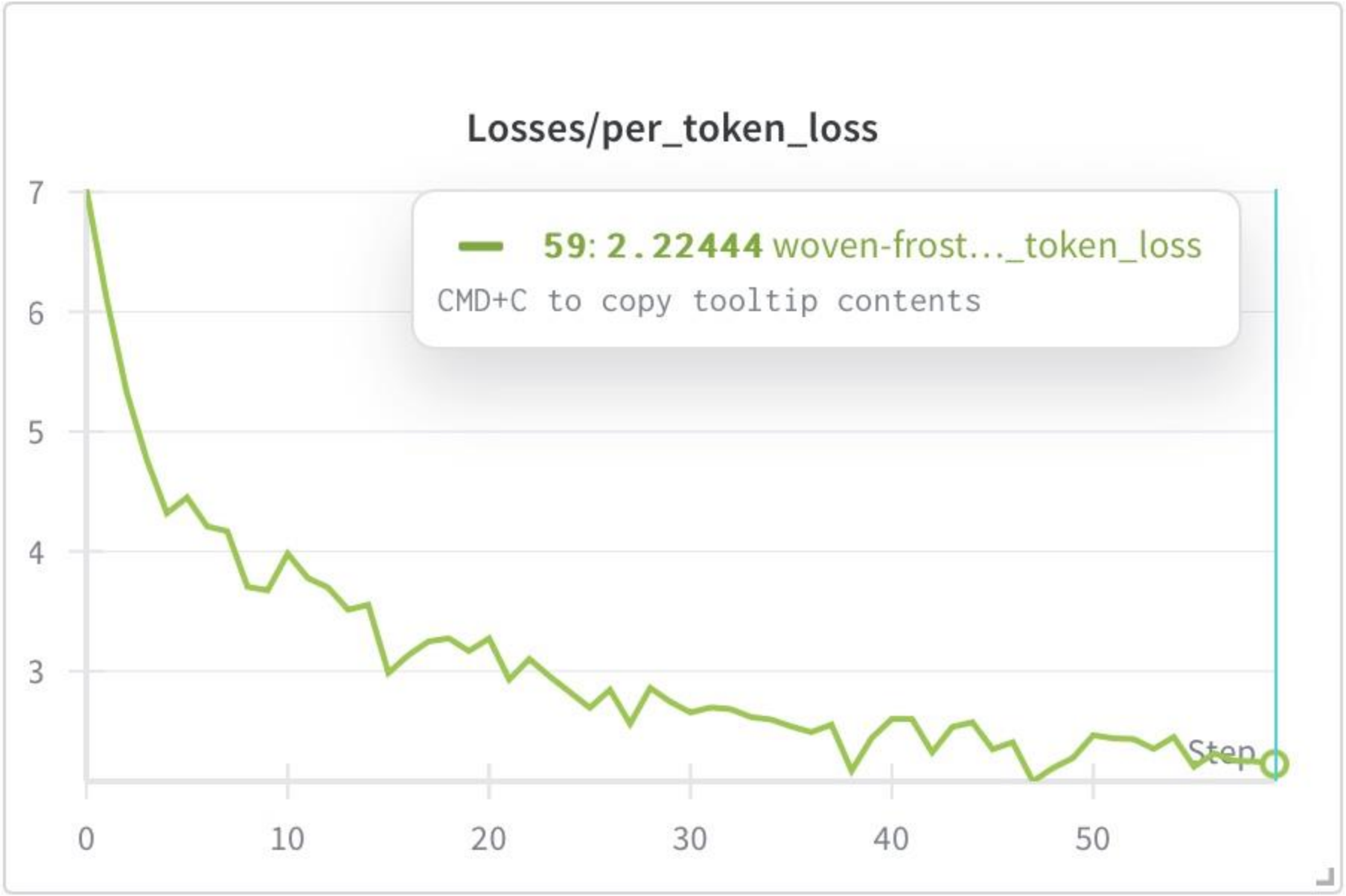
- **Perplexity Score:**
- For language models:

  Perplexity=e^{loss}
- Training stabilizing around **loss ≈ 2.2–2.5**.

  So:
- If loss = **2.2**

  PPL=$e^{2.2}$≈9.03
- If loss = **2.5**

  PPL=$e^{2.5}$≈12.18
- This is **very good** for a generative model trained on a specialized physics dataset.
- For comparison:
- GPT-2 small trained on WikiText-103:
  PPL ≈ 35
- Fine-tuned domain models:
  10–20



HVAE Metrics :

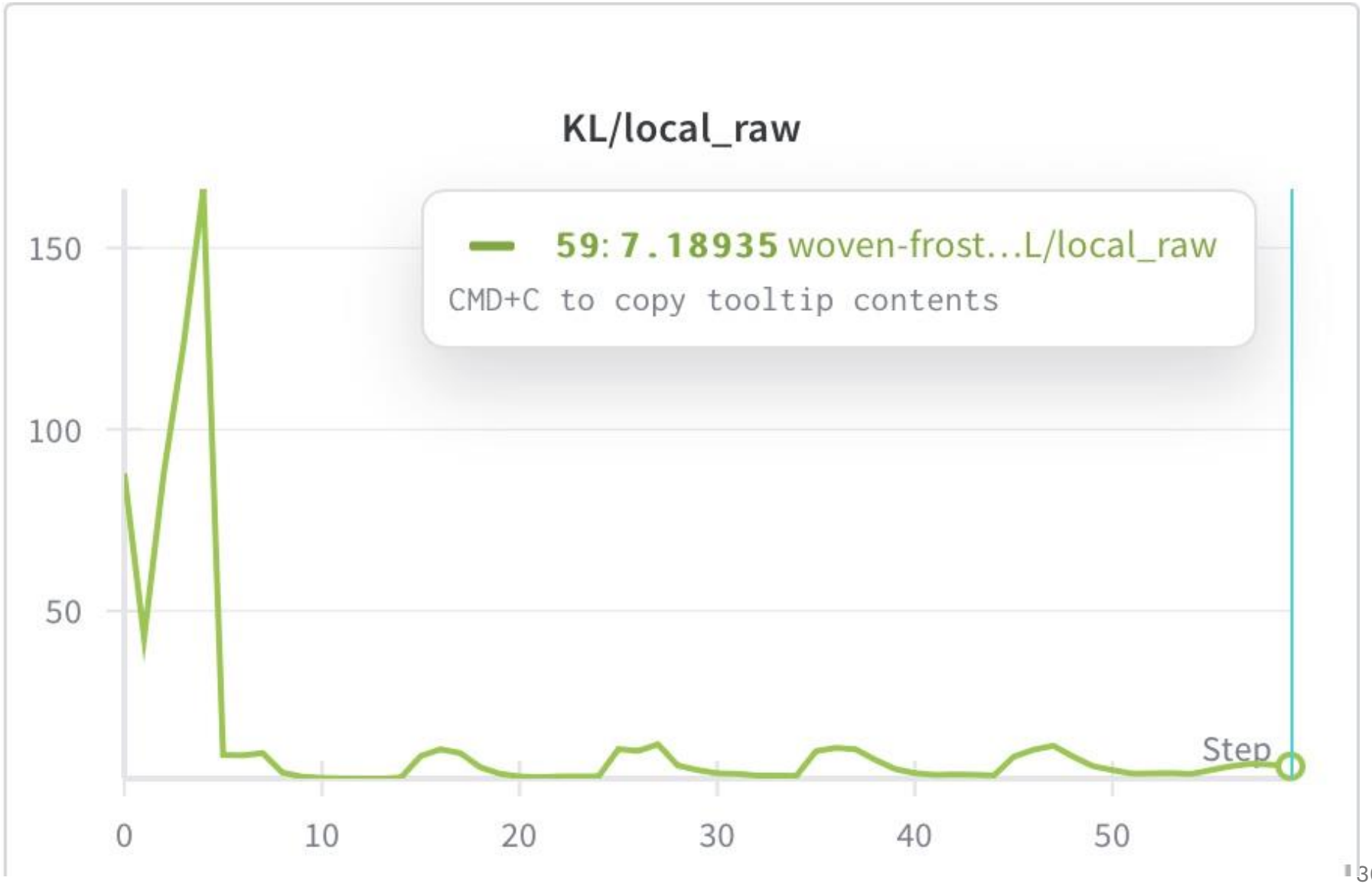
- We can infer:

- **BLEU Score** is likely **moderate**, because HVAE tends to generate *more diverse* phrasing.
- Depending on dataset, BLEU ≈ **20–35** is typical for domain-specific models.
- **ROUGE Score** is usually higher than BLEU in scientific datasets.
- Likely **ROUGE-L ≈ 40–55**, depending on whether inference involves reconstructing sentences.

- Hierarchical VAE uses **two levels of latent variables**:
- **Global latent vector**:
  captures large-scale structure or overall patterns
- **Local latent vectors**: capture fine-grained, per-token details
- To make sure the model **does not ignore its latent variables** (posterior collapse), we monitored:
  - KL divergences
  - Free-bits thresholds
  - Active units
  - Latent variances

HVAE Metrics : KL Local

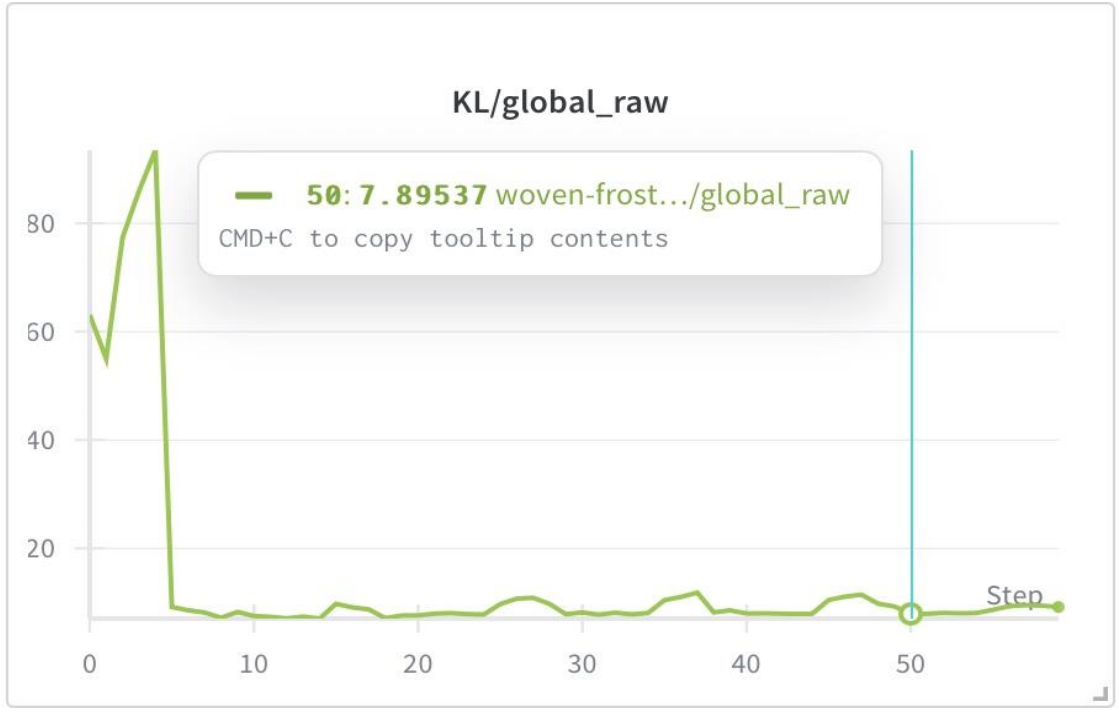
- Local KL = 7.189 (threshold was 4)
- We have this KL local loss for vector of 128 dimensions for the HVAE training.



HVAE Metrics : KL Global

- Free bits act like a “minimum learning pressure” on each latent group.
- They prevent the KL from collapsing to zero (which would mean the latent variables are being ignored).
- **Local KL > 4** ⇒ local latent variables are being used
- **Global KL** ≈ threshold then > 8 ⇒ global latent variables are also active

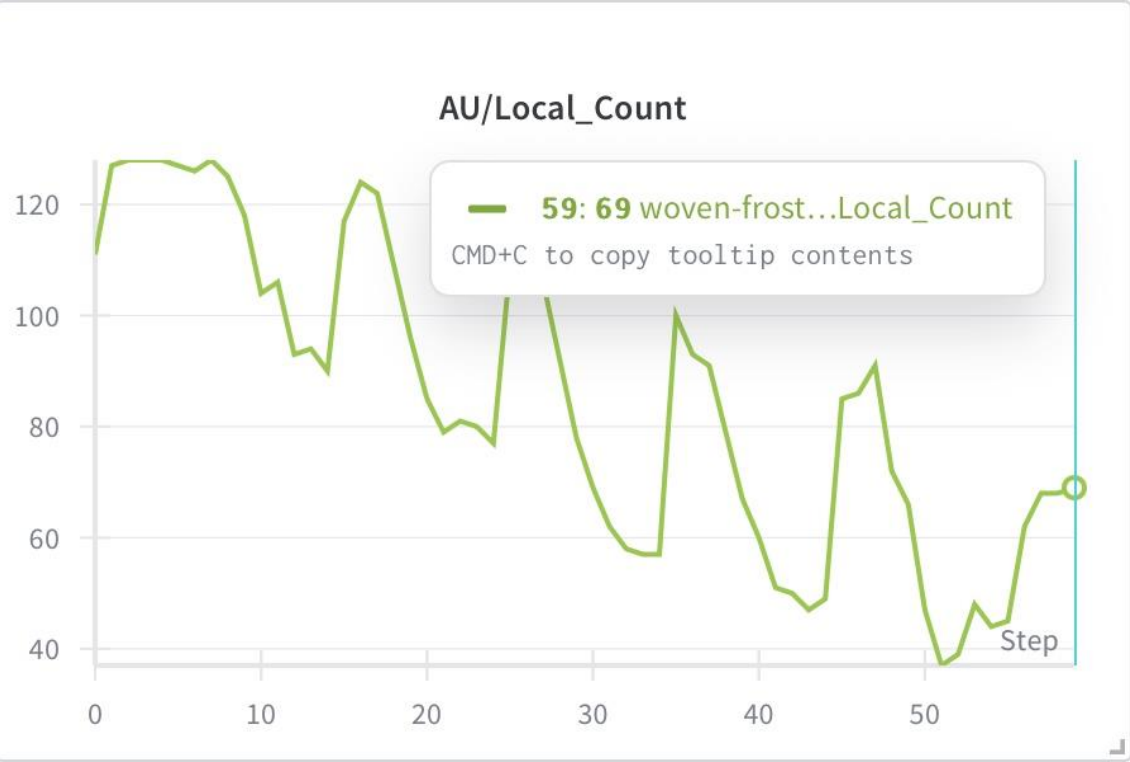
- Global KL = 7.895 at epoch 50 (threshold was 8)
- Global KL rises to 9.165 after β-squeeze at epoch 59.



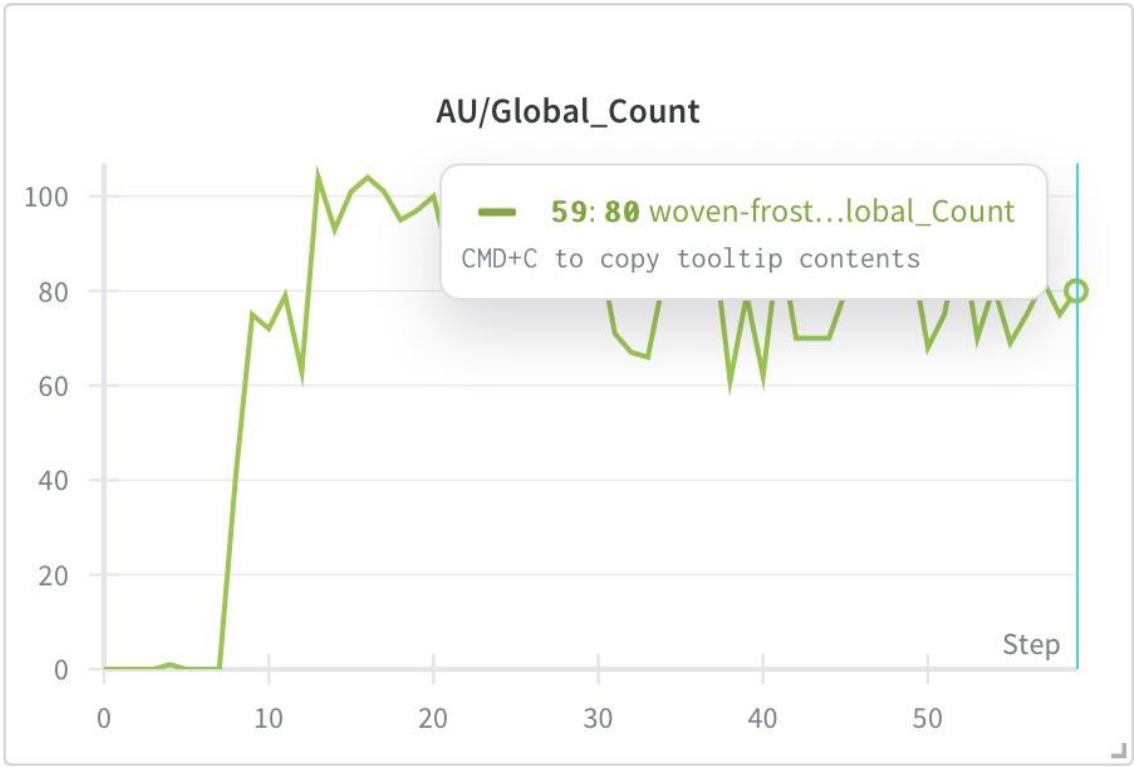
HVAE Metrics : KL Local and Global

- **Thoughts on β-squeeze and KL rise :**
- When β was reduced (toward the end of training),
- Lower β means “we care more about reconstruction now.”
  This allows the model to push more information into latents.
- The KL increase tells us:
- “Once the constraint was relaxed, the model naturally chose to store **more** information in the global latent vector.”
- This is **a good sign**. It shows that:
  - The model wants to use latent information
  - The hierarchical design is beneficial

HVAE Metrics : Active Units Local and Global



- Active units = number of dimensions that actually encode meaningful information (vari ance significantly different from 0).

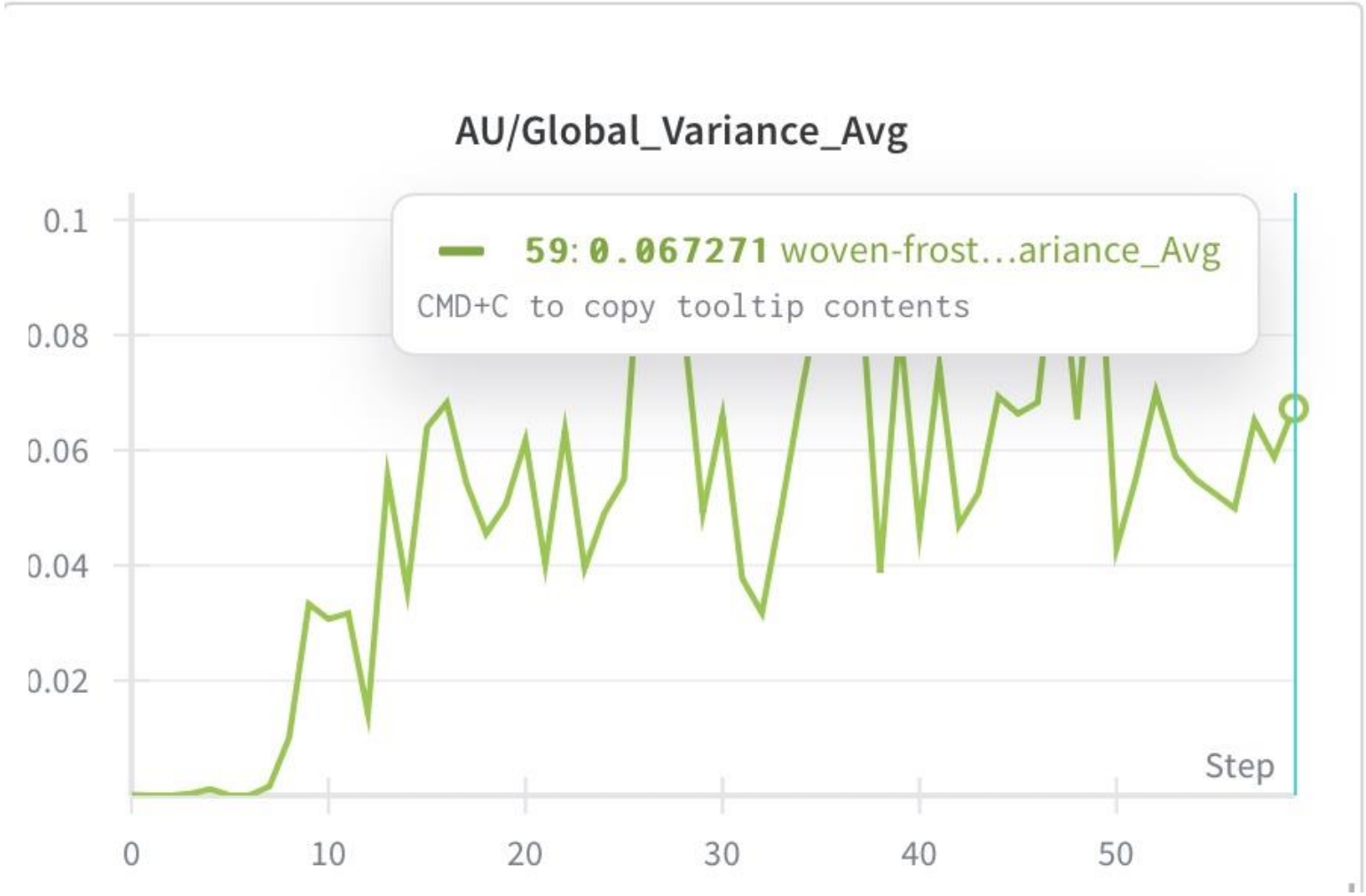


- Global latents: 80/128 active
- Local latents: 69/128 active
- Model uses global structure heavily (no collapse). Most of the important structure in the physics data is global rather than local. This shows that the hierarchical design is working correctly.”
- Fine-details still encoded locally (no collapse)

HVAE Metrics : Latent Variance

- Low-but-not-zero variance means:
- Latent dimensions are **not collapsed**
- Variance is concentrated in the **active dimensions** (about 0.1 each)

- The model learned compact, meaningful latent features without making them overly noisy or degenerate.



Questions?

STEVENS
INSTITUTE OF TECHNOLOGY
1870

THANK **YOU**

Stevens Institute of Technology
1 Castle Point Terrace, Hoboken, NJ 07030