🧠 What does this equation mean?

🔷 Equation:

$$p(H \mid z, P) = p(z \mid P) \cdot p(H \mid z, P)$$

It looks like it's repeating itself, but it's actually describing the **joint generation process** using a **latent variable** `z`.

Let's break it piece-by-piece:

🔍 1. What are P, H, z?

- `P` = Premise
  E.g., "I have a cat."
- `H` = Hypothesis
  E.g., "I own a pet." (This *should* follow from the premise)
- `z` = Latent entailment vector
  This is a learned abstract representation of the **relation** between `P` and `H` (like "entailment", "neutral", "contradiction").

🔄 2. The Probabilistic Meaning

Full Joint Distribution:

This is basically:

$$\underbrace{p(H \mid P)}_{\text{Overall Goal}} = \int_z \underbrace{p(z \mid P)}_{\text{Latent Relation}} \cdot \underbrace{p(H \mid z, P)}_{\text{Generate H}} dz$$

We can't directly model `p(H | P)`, so we **introduce** a latent variable `z` to factor it:

- First: guess the **relation type** `z` from `P`
- Then: generate `H` from both `P` and `z`

This technique is known as the **latent variable model** or **variational inference**.

🔧 3. ERM Mechanism Explained

Let's now walk through the process.

🟩 Step 1: Encode Premise

Input (with special tokens):

```
[z] [SOP] I have a cat [EOP]
```

Assume embeddings are 4-dimensional. Token embeddings (random dummy values):

```
e[z] = [0.2, 0.1, 0.0, 0.3] e[SOP] = [0.1, 0.1, 0.1, 0.1] e[I] = [0.5, 0.2, 0.1, 0.4]
e[have] = [0.6, 0.3, 0.3, 0.2] e[a] = [0.1, 0.2, 0.2, 0.1] e[cat] = [0.3, 0.4, 0.1,
0.5] e[EOP] = [0.2, 0.2, 0.2, 0.2]
```

So the sequence becomes a matrix:

```
E = [
 [0.2, 0.1, 0.0, 0.3],  # [z]
 [0.1, 0.1, 0.1, 0.1],  # [SOP]
 [0.5, 0.2, 0.1, 0.4],  # I
 [0.6, 0.3, 0.3, 0.2],  # have
 [0.1, 0.2, 0.2, 0.1],  # a
 [0.3, 0.4, 0.1, 0.5],  # cat
 [0.2, 0.2, 0.2, 0.2],  # [EOP]
]
```

🟧 Step 2: Pass into Encoder

This is passed to BART encoder, and the output of the `[z]` token gives:

```
h_z = [0.15, 0.1, -0.05, 0.2] # shape [4]
```

This is your **query** to memory `M`.

🟦 Step 3: Memory Lookup for Latent z

Suppose the memory `M` stores 3 latent entailment types:

python Copy Edit

```
M = [ [1.0, 0.0, 0.0, 1.0], # entailment [0.0, 1.0, 1.0, 0.0], # neutral [1.0, 1.0, 0.0, 0.0], # contradiction ]
```

Dot product of `h_z` with each row:

python Copy Edit

```
score_1 = 0.15*1 + 0.1*0 + (-0.05)*0 + 0.2*1 = 0.15 + 0.2 = 0.35 score_2 = 0.1 + (-0.05) = 0.05 score_3 = 0.15 + 0.1 = 0.25
```

Apply softmax:

python Copy Edit

```
softmax([0.35, 0.05, 0.25]) ≈ [0.41, 0.30, 0.29]
```

So the retrieved **latent z** is:

python Copy Edit

```
z = 0.41*M1 + 0.30*M2 + 0.29*M3
```

Now, `z` is your final **entailment vector**.

🟧 Step 4: Generate Hypothesis H

We feed `z` + the encoded P into the decoder to generate `H`.

So:

$$p(H \mid P) = \int_z p(z \mid P) \cdot p(H \mid z, P)$$

The decoder uses `z` as a **bias** or additive vector to condition generation (e.g., at `[SOH]` token).

📌 In Short

- `p(z | P)` = get latent relation using `h_z` as query to memory
- `p(H | z, P)` = decoder generates response using both
- Latent memory **learns relation types** (entailment, contradiction, etc.) during training
- Helps model avoid contradictions in persona