

CIS 6280 · FALL 2026

WORLD MODELS

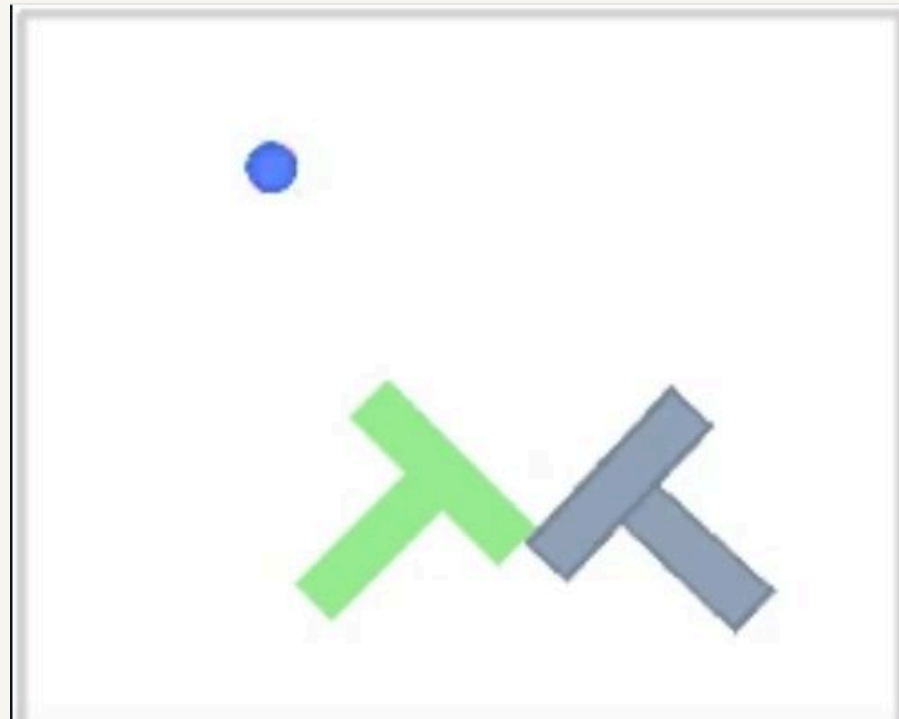
LECTURE 5

Representation Learning I

This Week: Representation Learning

Today · Sep 8	Representation Learning I: data targets and view matching.
Thursday · Sep 10	Representation Learning II: teachers, feature targets, and distribution constraints.
Course materials	Lecture pages and readings are linked from the course site.

A World Model Needs a Useful Description of the Scene



Blue: pusher. Gray: block. Green: target pose.

Object names

"A pusher and a T-shaped block."

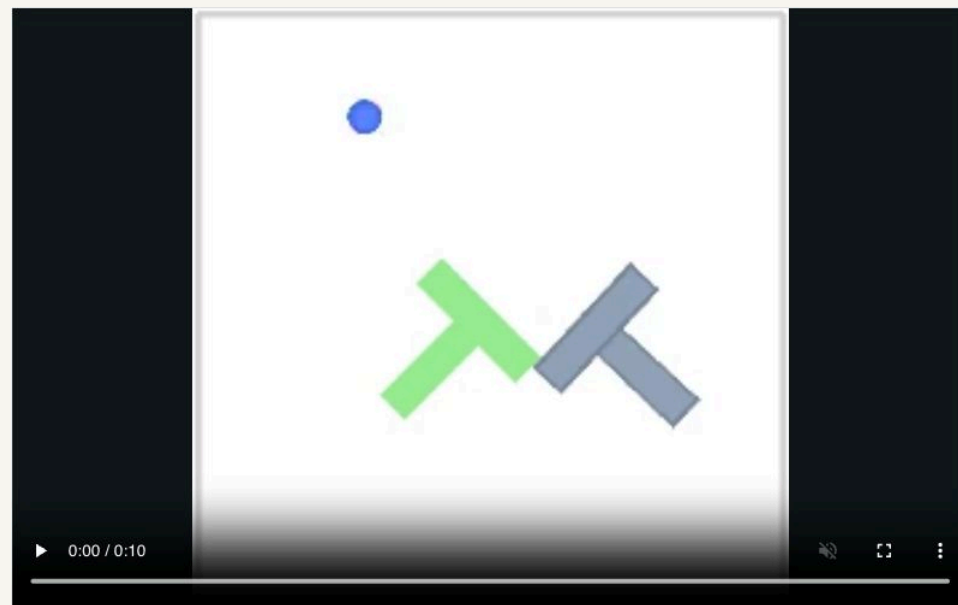
Spatial information

Where the pusher is. Where the block is.
How the block is turned.

What would you record to predict the result of the next push?

A useful description keeps the information the prediction needs.

A Representation Expresses Information in a Chosen Form



Pixels

An array of RGB values.

Measured coordinates

Object pose and pusher position.

Learned features

Numbers produced by an encoder.

$$e = E_{\phi}(o)$$

Learning the representation changes which relationships are easy to compute.

Observation Representations and Markov States

Representation z_t

$$z_t = E_\phi(o_t; o_{<t}, a_{<t})$$

Another form of observation information.

It can include history and motion.

No Markov assumption is required.

State s_t

$$s_{t+1} \sim p_\theta(\cdot \mid s_t, a_t)$$

The variable that evolves in the world model.

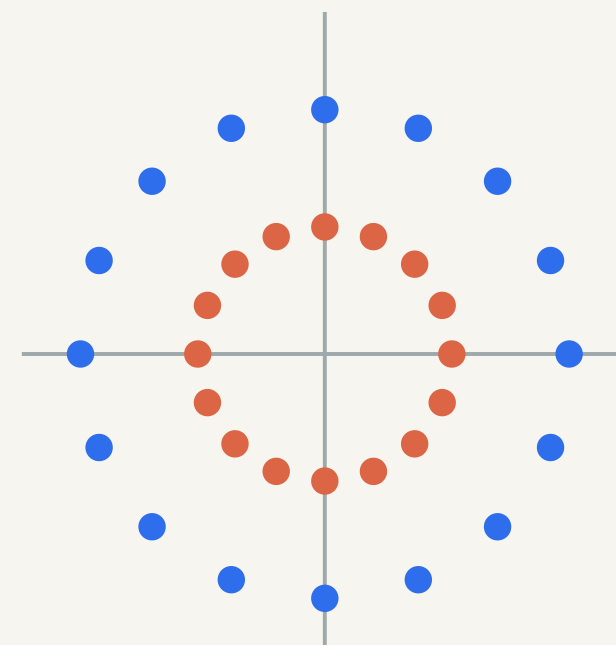
Given the current state and action, earlier states do not change the next-state distribution.

L4 assumes this evolution is Markov.

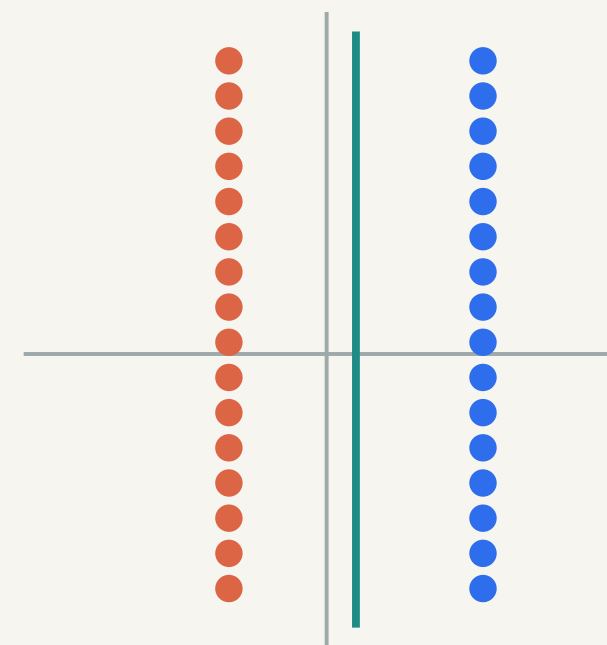
$$p(s_{t+1} \mid s_{0:t}, a_{0:t}) = p(s_{t+1} \mid s_t, a_t)$$

Using temporal context does not automatically make a representation a Markov state.

Coordinates Can Make a Simple Readout Work



Cartesian: (x, y)



Polar: (r, θ)

The class information is unchanged; a threshold on radius becomes sufficient.

How Do We Test Whether a Representation Is Good?

Example: read the object's orientation



Image → **frozen encoder** → feature e

Frozen = encoder weights stay unchanged.

Train a linear probe

$$\hat{y} = We + b, \quad y = (\cos \alpha, \sin \alpha)$$

1. Fit: train only W, b on labeled images.

Minimize squared error to the known orientation target.

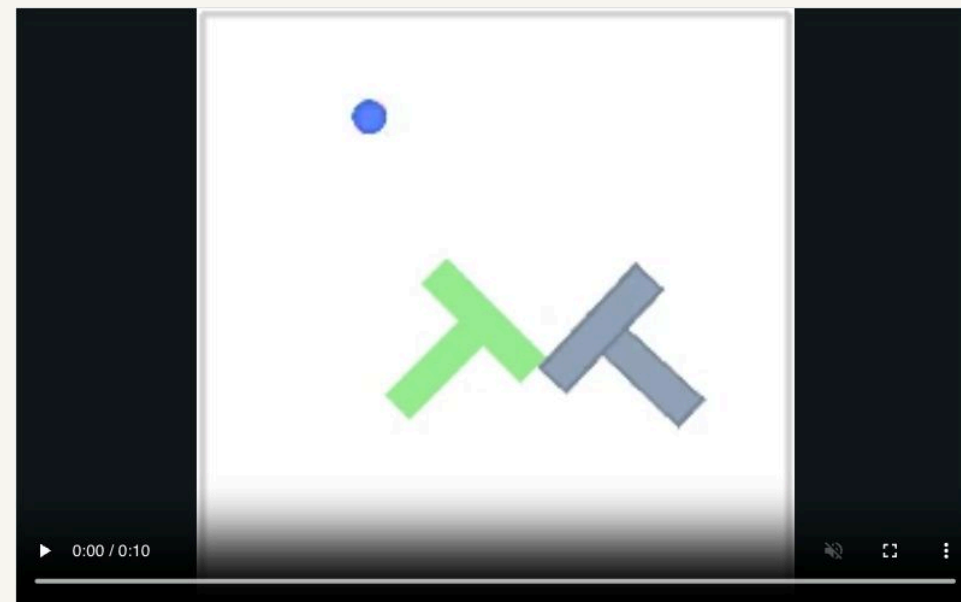
2. Test: predict angles in unseen scenes. Report angular error.

For classification, predict class scores and train with cross-entropy.

Compare encoders fairly: same labels, feature extraction, probe type, and test scenes.

A successful linear probe shows easy access. Failure does not prove the information is absent.

Global and Local Representations

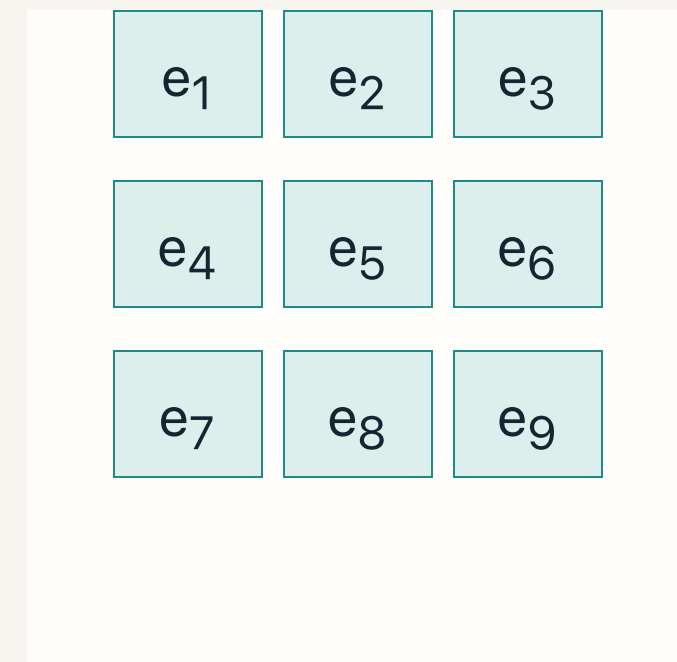


Global



One vector for the whole image.

Local



One vector for each image region.

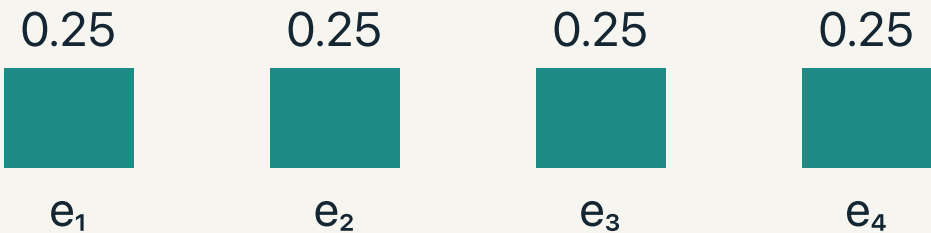
Local features keep their spatial arrangement and can use whole-image context.

Linear and Attention Probing Read Features Differently

Both freeze the encoder. Start from the same patch features $e_1, \dots, e_K$.

Linear probing

Use fixed pooling, then a linear head.



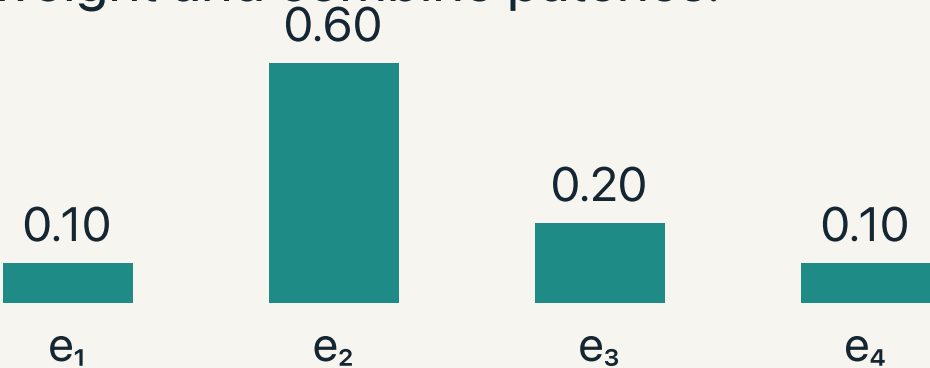
$$\hat{y} = W \left(\frac{1}{K} \sum_k e_k \right) + b$$

Train: only the final W, b .

Mean pooling is one choice. A fixed CLS feature is another.

Attention / attentive probing

Learn how to weight and combine patches.



$$\hat{y} = W \left(\sum_k \alpha_k v_k \right) + b$$

Train: attention pooling + final head.

A learned query scores each patch. Weights depend on the input; v_k are projected features.

Illustrative weights, not measurements. Fit both probes with task labels. Test on the same held-out data.

Attentive probing learns a richer readout. Report the probe type when comparing scores.

Features Provide Inputs for Future Prediction

Observations

$O_0, \dots, O_t$

Encoder E_ϕ



Features

$z_0, \dots, z_t$

Predictor F_θ



Next observation

$\hat{O}_{t+1}$

The predictor also receives known actions. $\hat{O}_{t+1} = F_\theta(z_{\leq t}, a_{\leq t})$

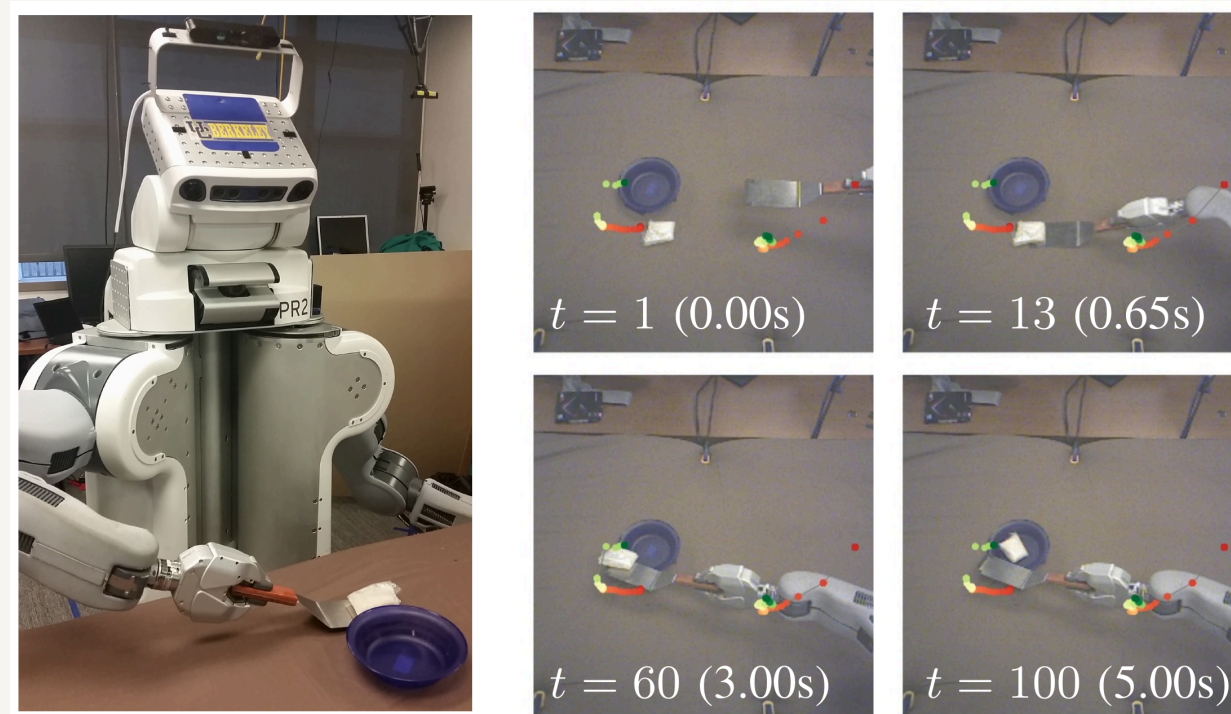
This week: learn the encoder and test its features.

Later: learn the predictor from sequences.

The predictor can use earlier representations when the latest feature is insufficient.

Robot Representations Need Spatial Information

Represent an image with point locations

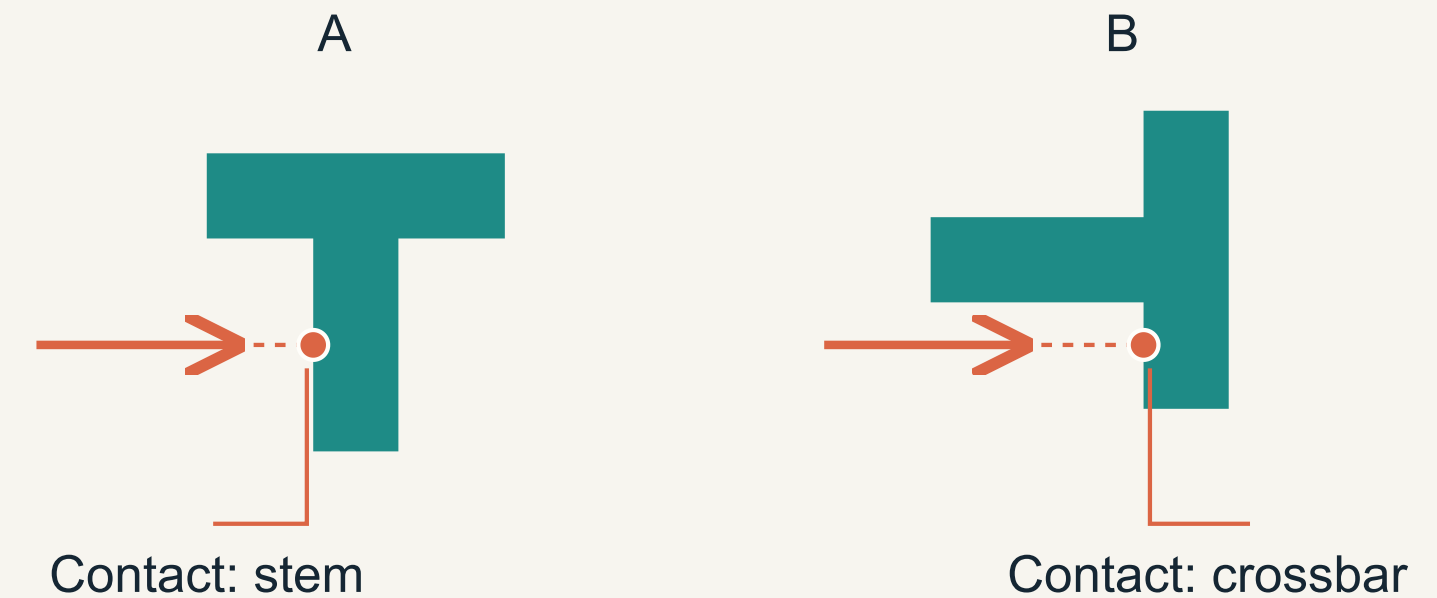


Colored marks are learned visual points.

$$e = (x_1, y_1, \dots, x_K, y_K)$$

Why do location and orientation matter?

Same T-shaped object. Same push to the right.



Different contact locations can lead to different motion.

Classroom schematic · orange dots mark first contact.

For prediction, preserve the positions and orientations that affect contact.

Whether Lighting Matters Depends on the Target

Downstream question	Lighting change	Desired behavior
Where is the object relative to the pusher?	Same geometry, different illumination	Keep geometry easy to read
What will the next RGB frame look like?	Brightness and shadows affect pixels	Retain or model appearance
What object is present?	Many lighting changes preserve identity	Stable category information

“Nuisance” is a relationship between a variation and a task.

Invariance and Equivariance Make Different Promises

Invariance

$$E(To) \approx E(o)$$

Ignore a transformation for a chosen readout.

Equivariance

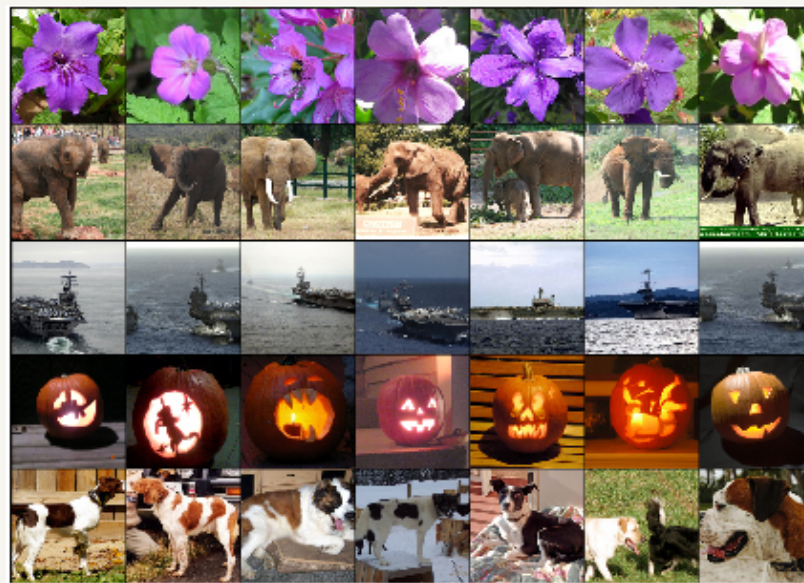
$$E(To) \approx \rho(T)E(o)$$

Track a predictable change in the representation.

A rotated object can keep its identity while its orientation changes.

Training a Neural Network Also Learns Its Representations

ImageNet classification

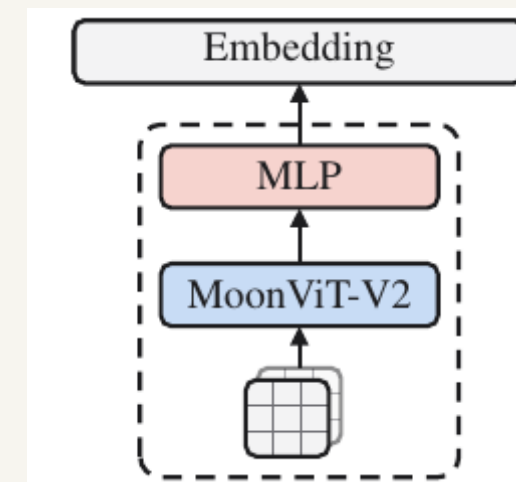


Training signal: human category labels.

Image → learned features → predicted class.

After training: nearby hidden features retrieve related images. AlexNet, Fig. 4.

A native MLLM: Kimi K3



MoonViT-V2

Visual features learned from scratch through next-token prediction.

Training signal: text to predict given image + text context.

Image → visual features; features + text → next text token.

The loss trains the vision encoder, projector, and language backbone together.

Training updates both the output head and the layers producing its features.

Useful representations can be learned while solving another task.

Rewards and Future Observations Can Also Teach Features

AlphaGo: learn to choose and evaluate moves



Board $\rightarrow$ learned features $\rightarrow$ policy / value.

Training signal: expert moves, then self-play and game outcomes.

The original AlphaGo combines supervised learning and reinforcement learning.

A world model: learn to predict observations



Observation history + actions $\rightarrow$ learned features $\rightarrow$ future prediction.

Training signal: what is actually observed next.

If trained jointly, prediction error also updates the observation encoder.

Representation learning occurs in supervised learning, RL, and predictive modeling.

Can We Learn from Experience Without Labeling Every Moment?



Look, act, observe what changes, and interact.

World models need more than object names

What matters changes over time: pose, motion, contact, occlusion, and action consequences.

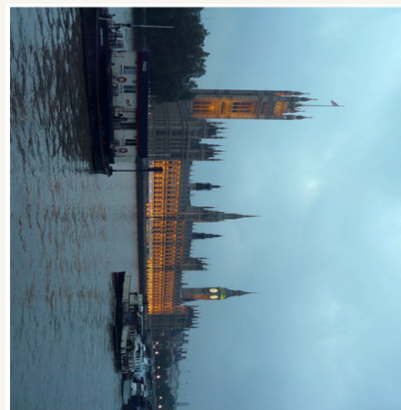
Annotating every detail in every frame is costly. Rewards may also be sparse or tied to one task.

Experience already contains relationships: neighboring moments, different views, and the outcomes of actions.

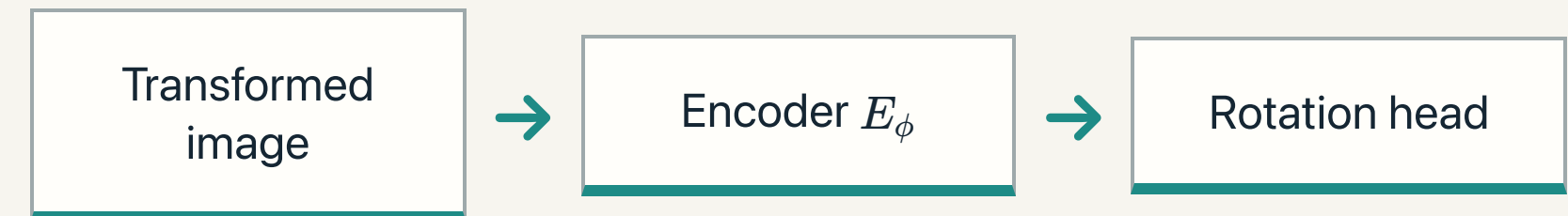
Human learning also includes language, instruction, and social feedback. The photo motivates the question; it does not identify a single learning algorithm.

This week: learn reusable representations from data without manual labels for every training target.

Self-Supervised Learning Uses Data to Create Training Targets



Apply a sampled 90° rotation



The sampler knows the answer: 90°.

$$\mathcal{L} = \text{CE}(q_\psi(E_\phi(T_{90^\circ}o)), 90^\circ)$$

After pretraining: keep the encoder and fit a new readout for angle or geometry.

Three Common SSL Families Organize the Learning Signal

Family	What supplies the training answer?	Examples
Reconstruction	Recover original content or structure from partial / altered data	Rotation · Jigsaw · DAE · MAE · AIM
Contrastive learning	Which other view belongs with this example?	SimCLR
Match learned targets	A teacher feature, assignment, or another view’s embedding	DINO · data2vec · JEPA

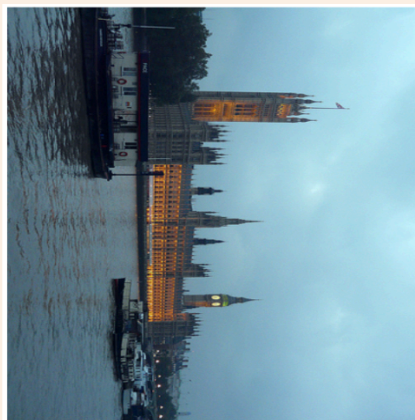
Across families: identify the visible input, target, loss, and reused features.

Reconstruction-Based Learning

Alter or hide information → predict the original content or structure.

These are **pretext tasks**: training tasks constructed to learn reusable features.

Rotate → recover orientation



Target: rotation angle.

The angle tells us how to undo the rotation.

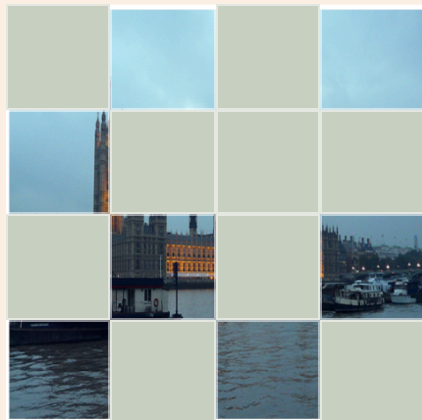
Shuffle → recover layout



Target: permutation class.

The permutation identifies the original layout.

Mask → recover content



Target: original pixels.

The original image supplies the missing values.

Altered input → encoder → features → predict the original information

Structure recovery uses angle / layout labels; data reconstruction predicts pixels or tokens.

The Patch Sampler Knows Their Relative Position



Reference patch



Query patch

Where was the query relative to the reference?

Right. The sampler records the offset.

Two patches



Shared encoder



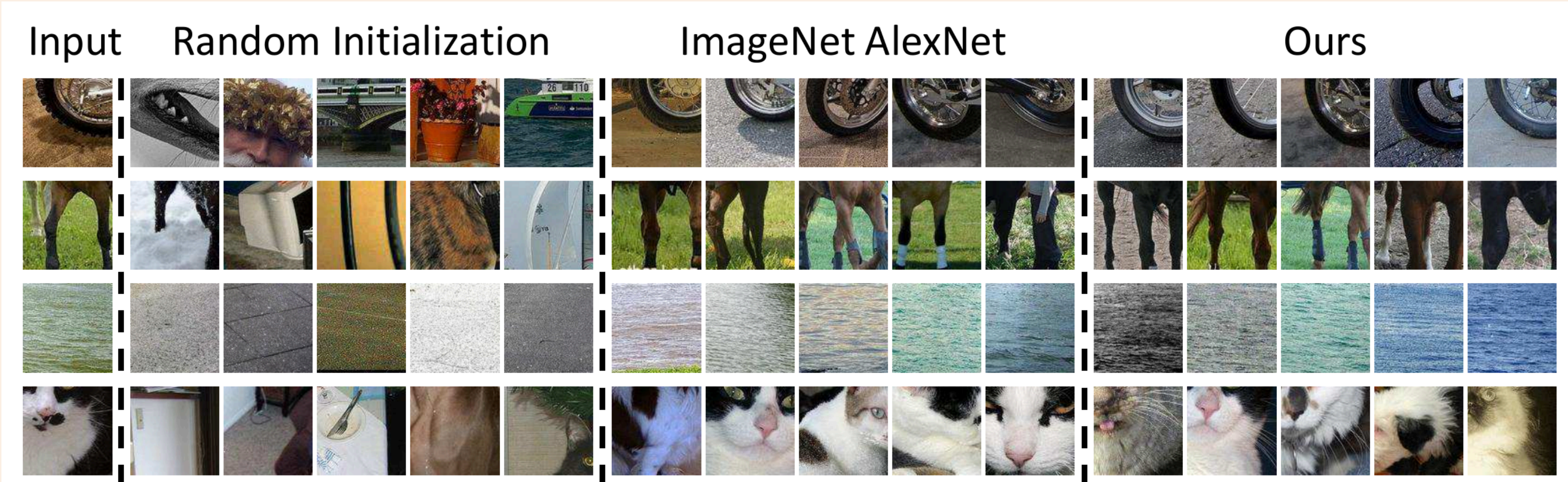
Pair classifier



Eight possible offsets

Recognizing parts and their arrangement can help solve this task.

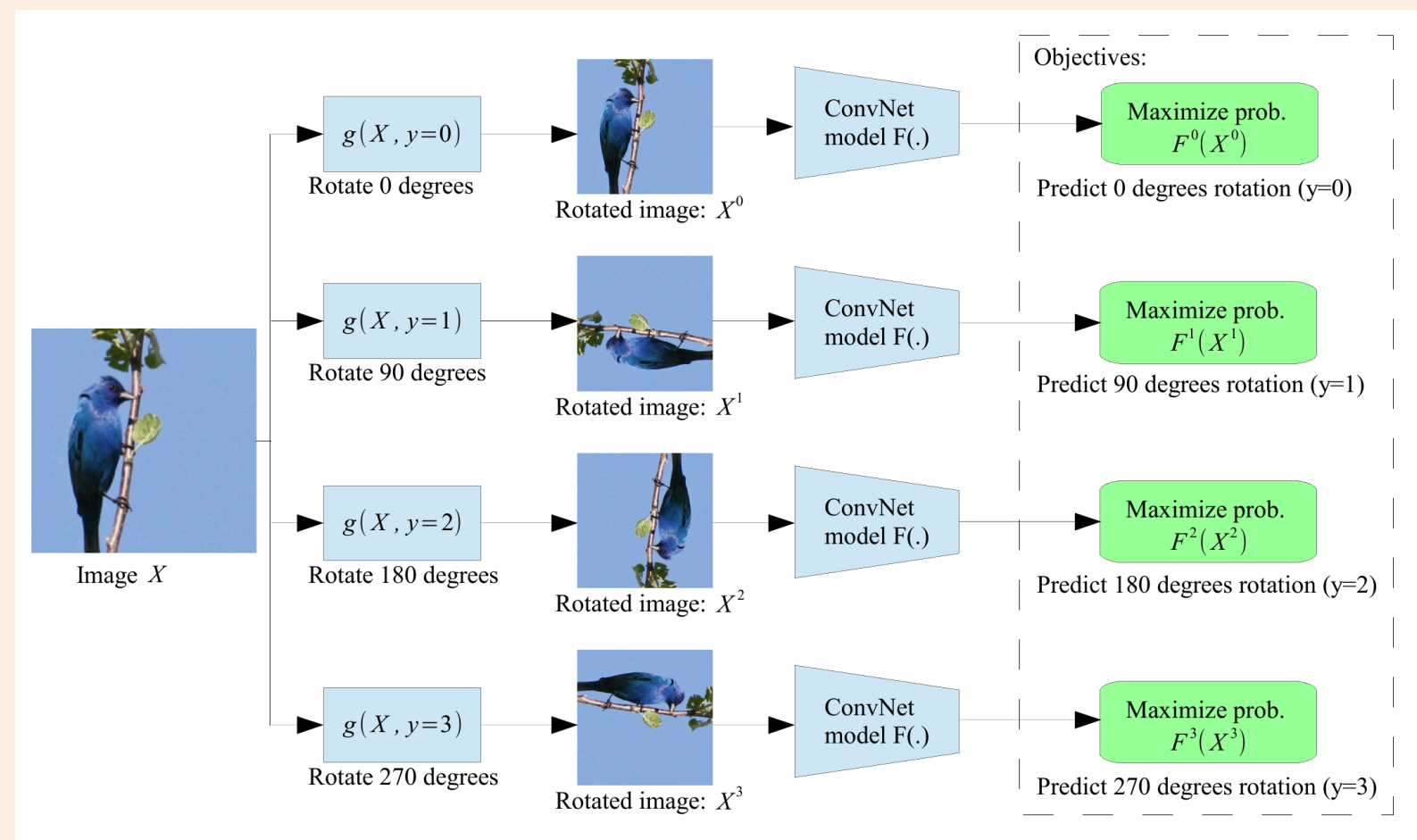
Position Prediction Can Produce Semantic Patch Neighbors



Read one row: query → random features → supervised features → context-trained features.

A new task—retrieval—reveals structure beyond predicting relative position.

RotNet Predicts the Rotation Applied to an Image

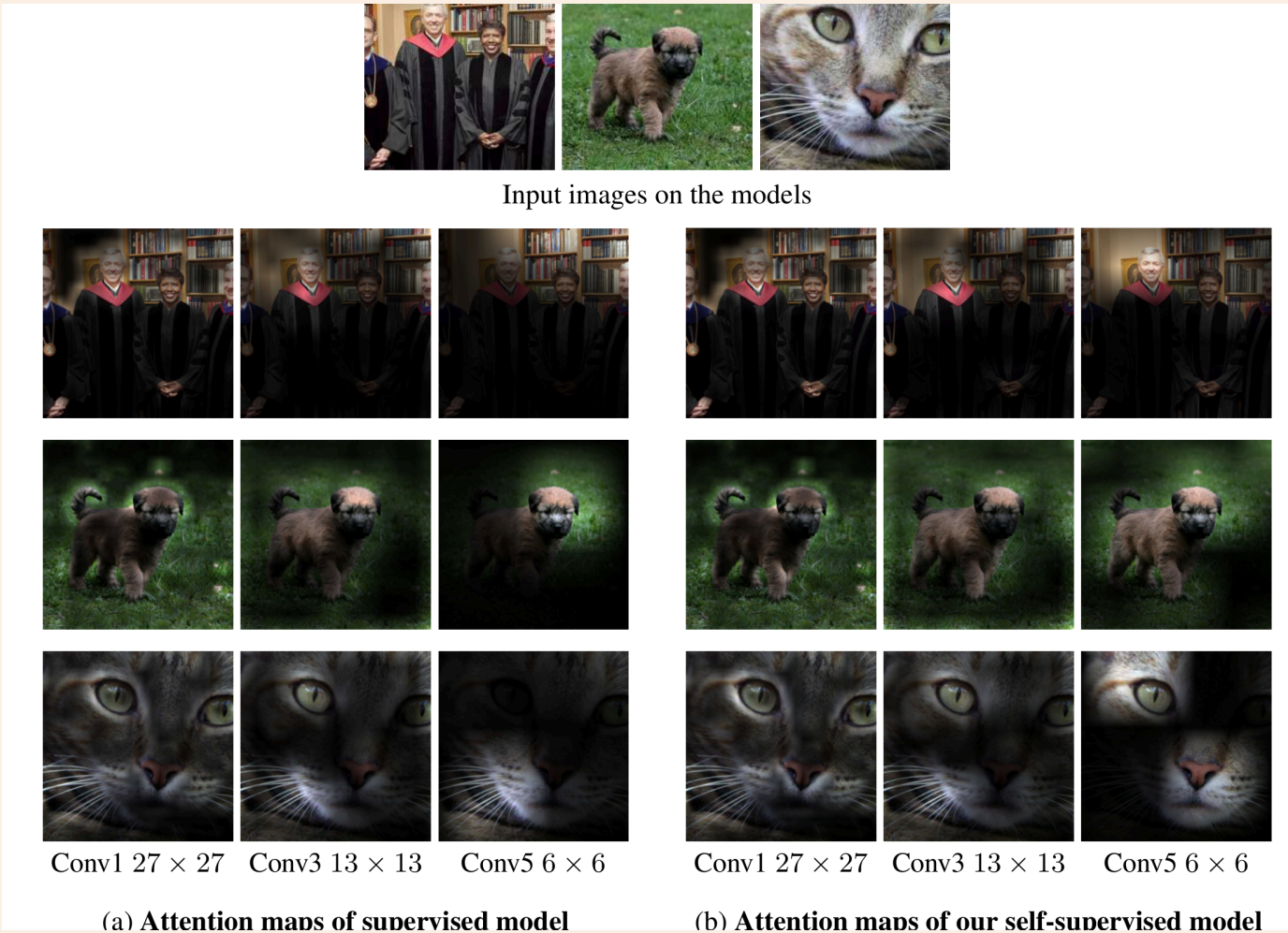


We generate the target: 0°, 90°, 180°, or 270°.

The encoder must retain cues that distinguish these transformations.

Predicting rotation and matching rotated views impose different requirements.

Rotation Training Changes Where Features Respond



Left group: supervised object recognition.
Right group: rotation prediction.

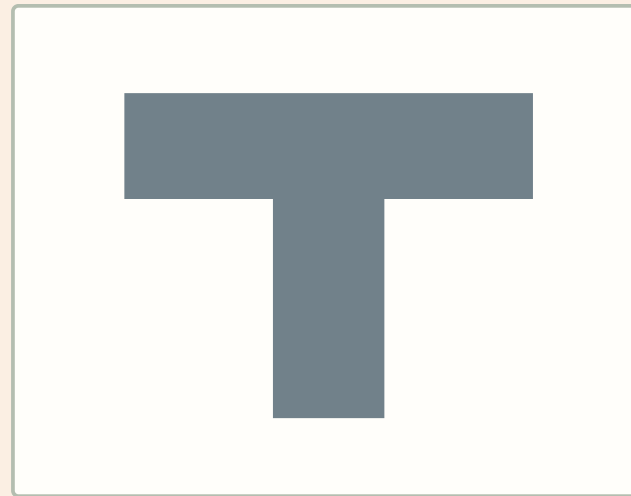
Compare the cat across network depths.

The maps suggest object structure; a separate probe must test the variable we need.

A Useful Task Hides the Answer but Preserves Clues

Task: restore the original shape, T or L, from the visible input.

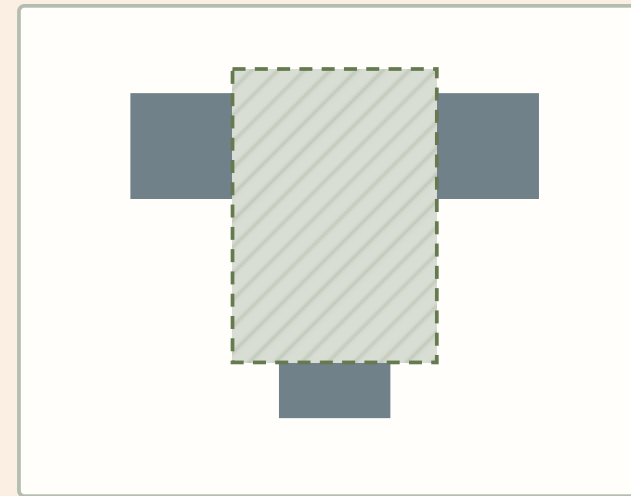
1. Entire shape visible



Copying is enough.

Low error need not require understanding the whole shape.

2. Middle hidden



Structure helps fill the gap.

The visible parts and their arrangement provide clues.

3. Entire shape hidden



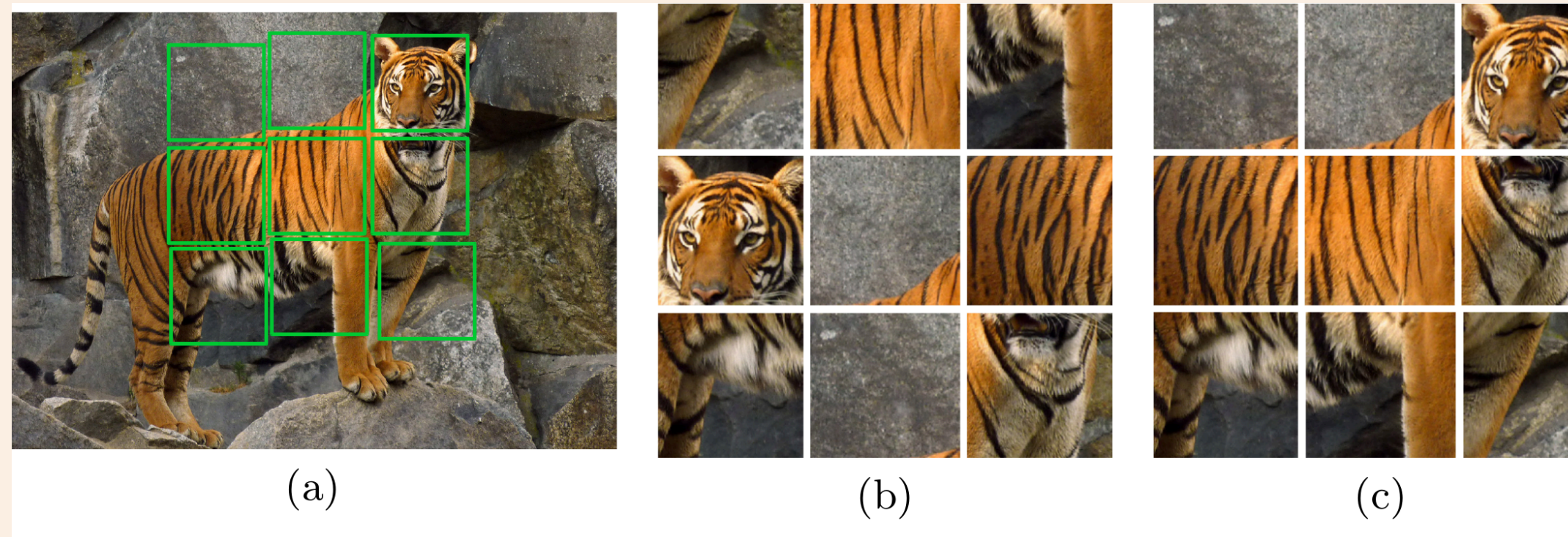
T or L? The input cannot tell.

More difficulty can mean more guessing, rather than better features.

Why can useful features emerge? When structure helps predict the hidden content, reducing error rewards features that encode that structure.

Remove the copying shortcut while retaining enough evidence to make inference useful.

Jigsaw Prediction Turns Spatial Layout into a Target



Predict which sampled tile permutation was applied.

Tiles with touching edges can expose a shortcut.

The training task should reward useful relationships between image regions.

Better Jigsaw Accuracy Can Mean Worse Transfer

Keep normalization and color jitter fixed	Jigsaw accuracy ↑	VOC detection mAP ↑
No gap between tiles	98%	47.7
Use a gap	88%	52.6

Gap	Normalization	Color jittering	Jigsaw task accuracy	Detection performance
✗	✓	✓	98	47.7
✓	✗	✓	90	43.5
✓	✓	✗	89	51.1
✓	✓	✓	88	52.6

Removing an easy cue hurts the puzzle score and improves detection in this experiment.

Different Training Tasks Encourage Different Features

Task	Constructed target	Possible useful evidence
Relative position	Neighbor class	Part layout and scene geometry
Rotation	Applied angle	Object orientation and upright structure
Jigsaw	Patch permutation	Spatial arrangement
Colorization	Missing color channels	Object and material context
Inpainting	Missing image region	Shape and contextual completion

Colorization Uses Context to Predict Missing Channels

Author examples: grayscale → predicted color

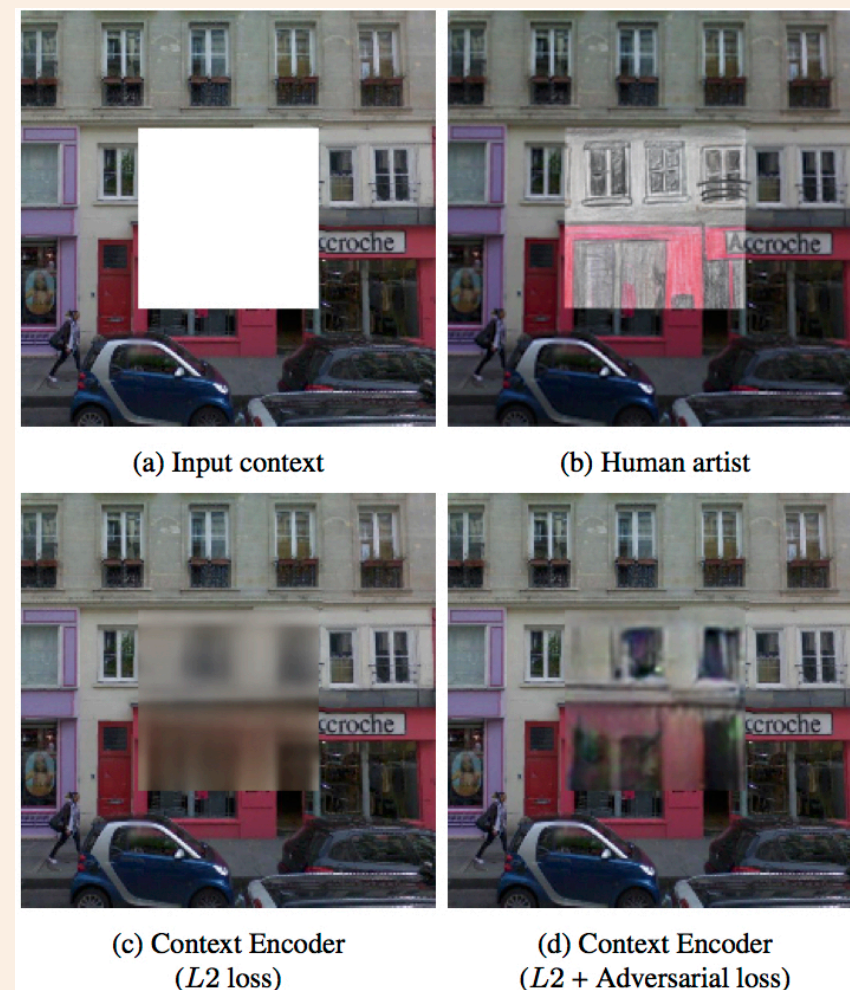


A failure case (original column labels)



The same grayscale evidence can support several plausible colors.

Inpainting Makes Context Useful for Filling a Hole



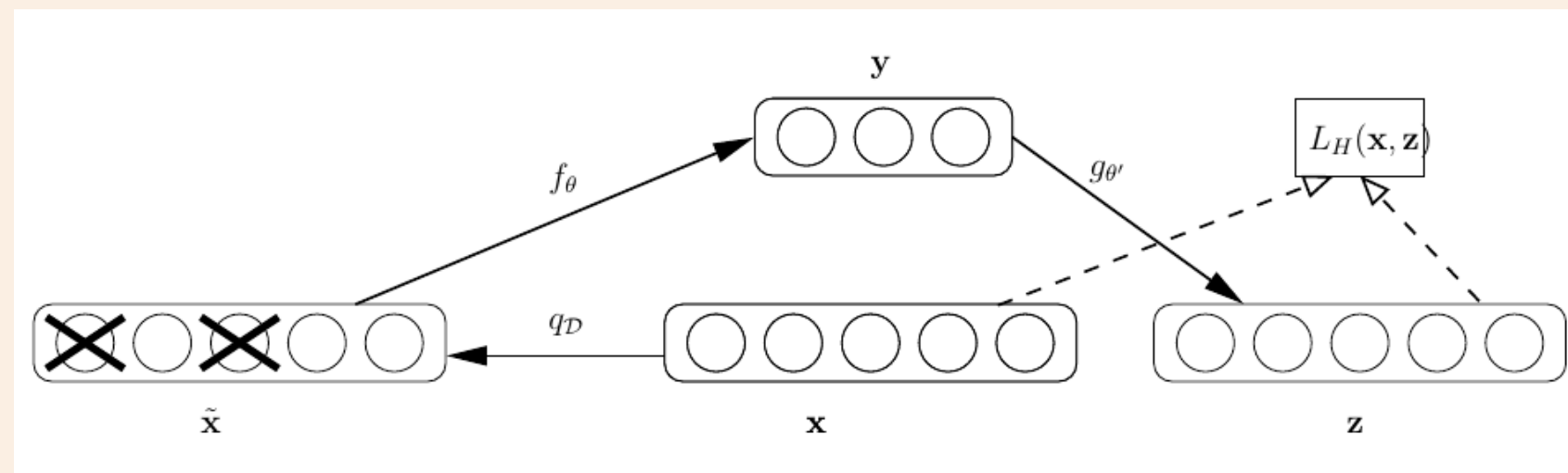
The missing center must be inferred from surrounding evidence.

Compare the two model outputs in the bottom row.

The loss changes which errors the reconstruction is encouraged to avoid.

Denoising Autoencoders Learn to Recover Clean Inputs

Corrupt the input; keep the clean original as the training target.



Original training diagram · Vincent et al. (2008)

The loss compares the reconstruction with **clean x**.

1. Corrupt $x \rightarrow \tilde{x}$: remove some values.
2. Encode $y = f_\phi(\tilde{x})$: learn features.
3. Decode $z = g_\theta(y)$: predict the clean input.

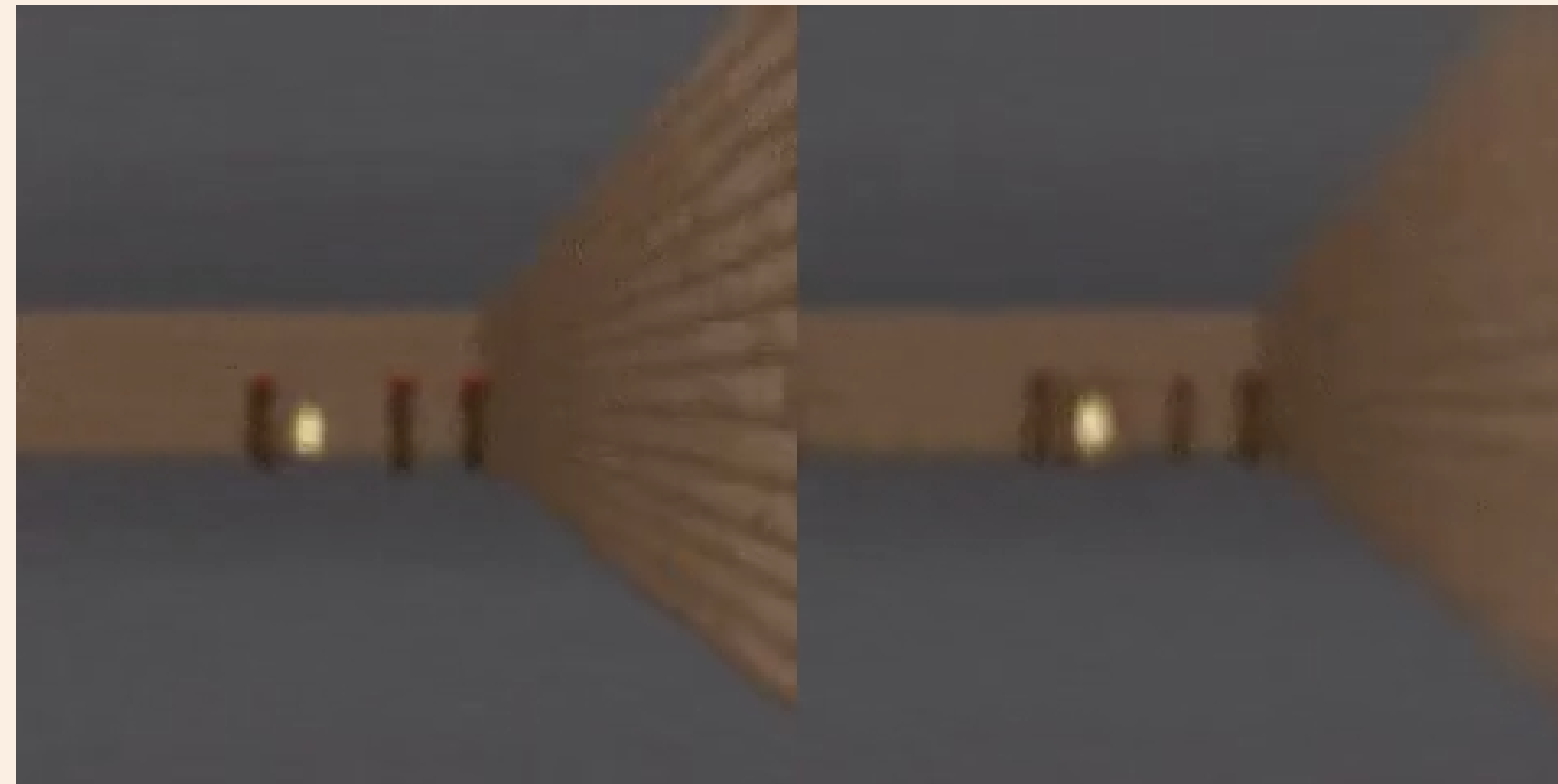
Training updates the encoder and decoder.

Other tasks can reuse the encoder's features.

$$\mathcal{L}_{\text{DAE}} = \mathbb{E}_{x, \tilde{x}} [\ell(g_\theta(f_\phi(\tilde{x})), x)]$$

Copying corrupted values is insufficient: recovery must use dependencies in the data.

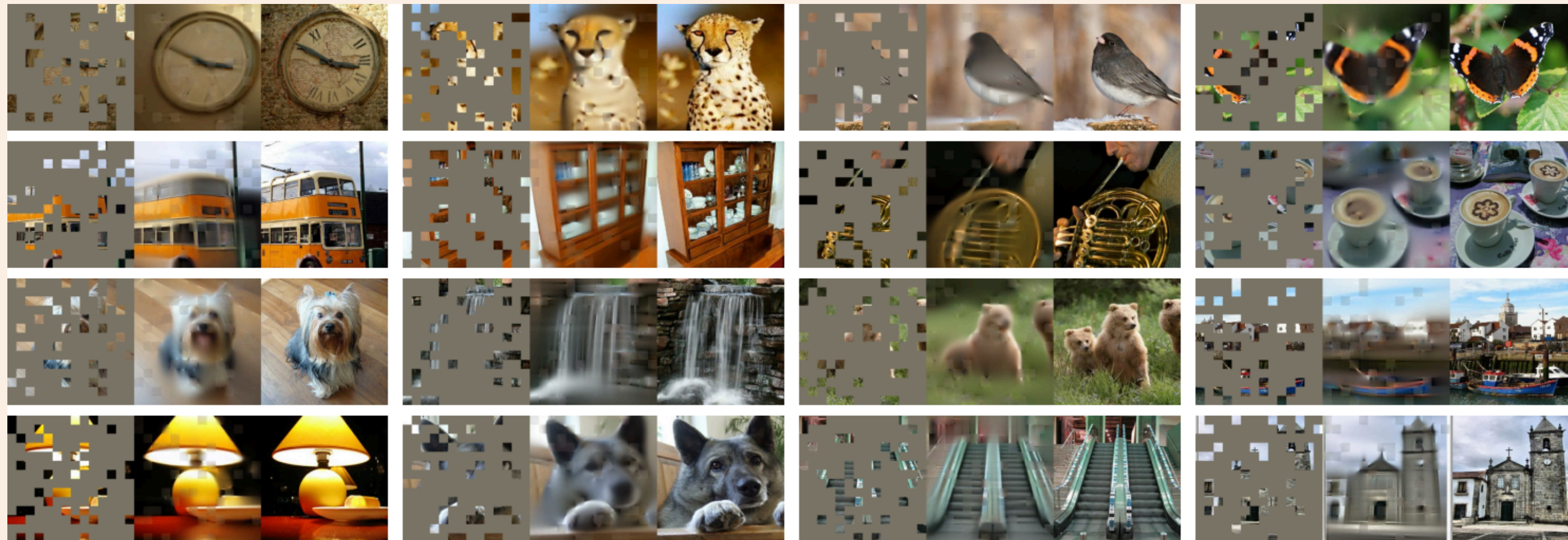
A Decoded View Makes Lost Detail Visible



Left: observed frame. Right: reconstruction through the visual code.

Pick one task-relevant detail, then propose a measurement beyond visual similarity.

Masking Removes Most of the Image from the Encoder



Author examples use 80% masking; the main training recipe uses 75%.

A Plausible Reconstruction Can Differ from the Observed Image



Each triplet: masked input → reconstruction → original. These examples use 80% masking.

Recovering a plausible scene and preserving the exact observed detail are different tests.

MAE Encodes the Visible Patches and Restores Their Positions



4 of 16 patches visible

1 • Encoder

Only the four visible patches enter.

2 • Decoder

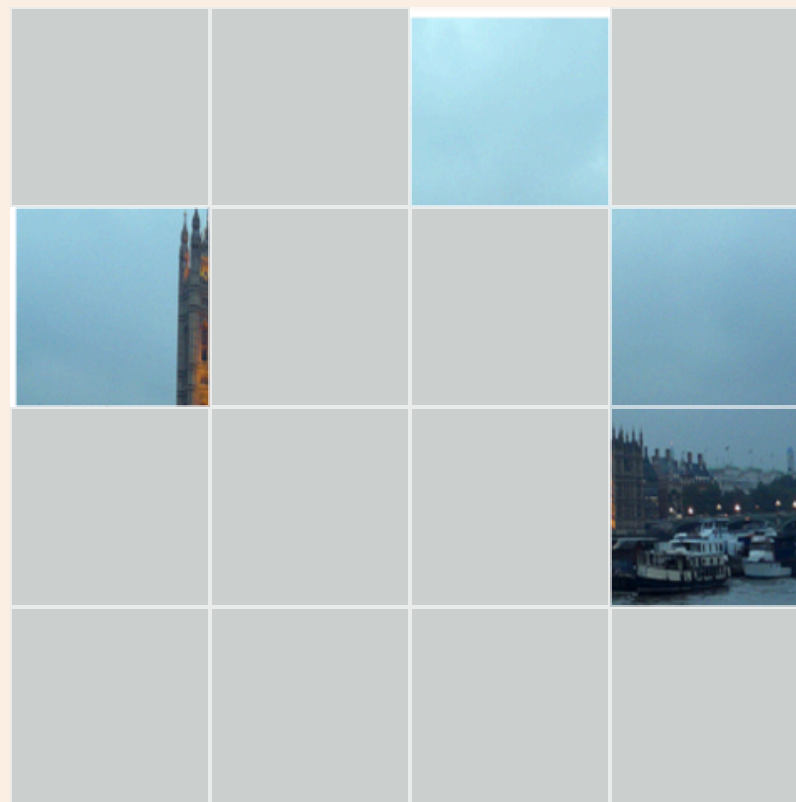
Visible features + mask tokens + all positions.

3 • Pixel prediction

Predict pixels; score the 12 masked patches.

Mask tokens enter the lightweight decoder, after the expensive encoder.

Masking Controls How Much Evidence the Encoder Receives



Masked patches

12 / 16 (75.0%)

Encoder receives 4 visible tokens.

Change only the mask. Which details become unavailable?

Resample mask

Visible tokens go to the encoder; mask tokens are inserted later for the decoder.

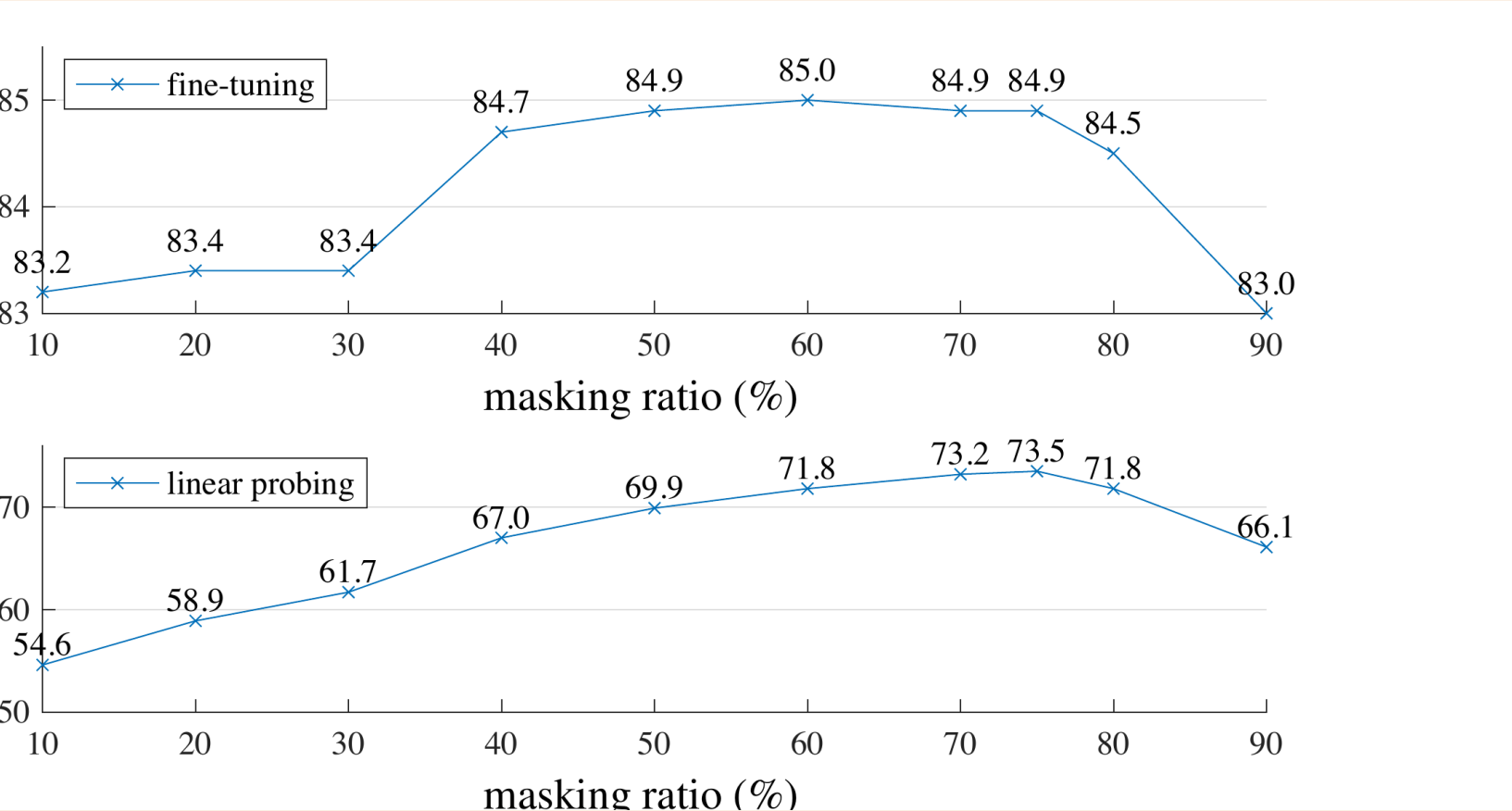
Only Masked Positions Contribute to This Reconstruction Loss

Patch index k	1	2	3	4
Original scalar target	1	2	3	4
Encoder sees	1	—	3	—
Decoder predicts	$\hat{x}_1$	$\hat{x}_2$	$\hat{x}_3$	$\hat{x}_4$
Scored here	No	Yes	No	Yes

$$\mathcal{T} = \{2, 4\}, \quad \mathcal{L} = \frac{1}{|\mathcal{T}|} \sum_{k \in \mathcal{T}} \|\hat{x}_k - x_k^{\text{target}}\|_2^2$$

Visible patches supply context; masked patches supply the scored answers.

MAE's Mask Ratio Is an Experimental Design Choice

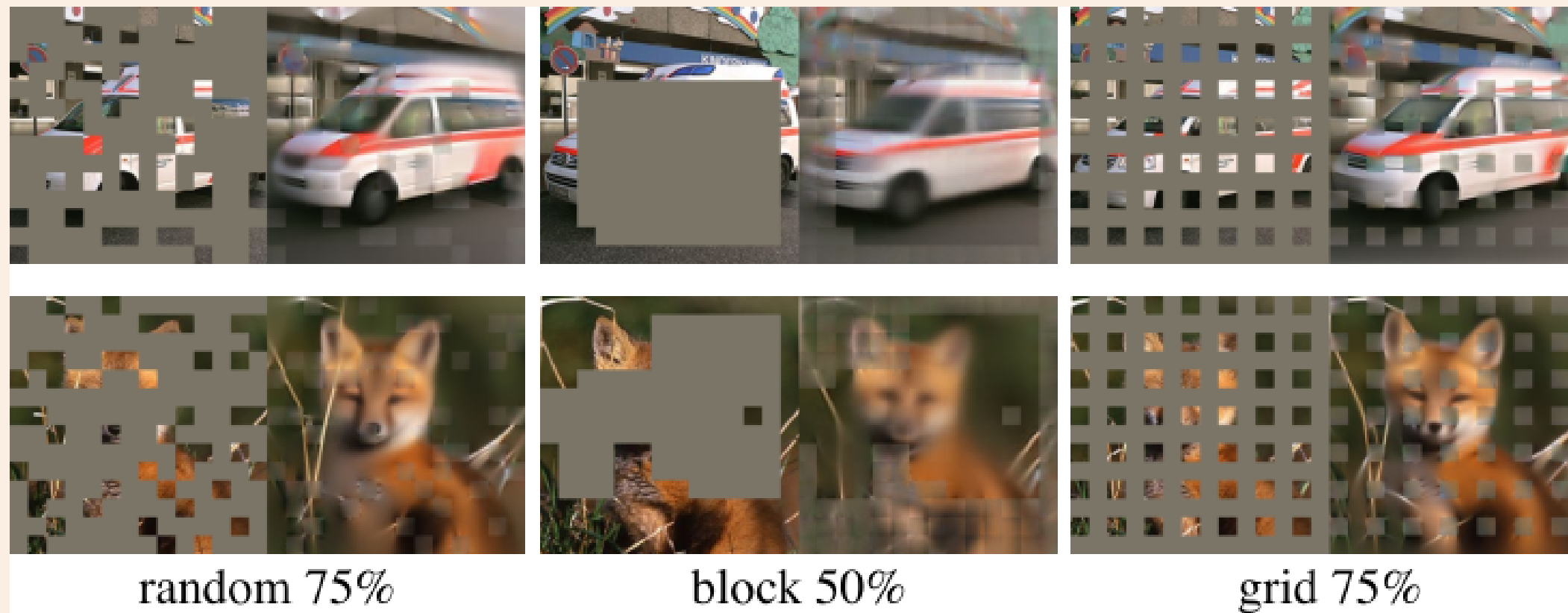


Top: update the whole encoder.
Bottom: freeze it and fit a linear readout.

Where is the useful range? Why do the two curves differ?

A strong result after fine-tuning does not imply equally accessible frozen features.

Mask Geometry Changes Which Clues Remain



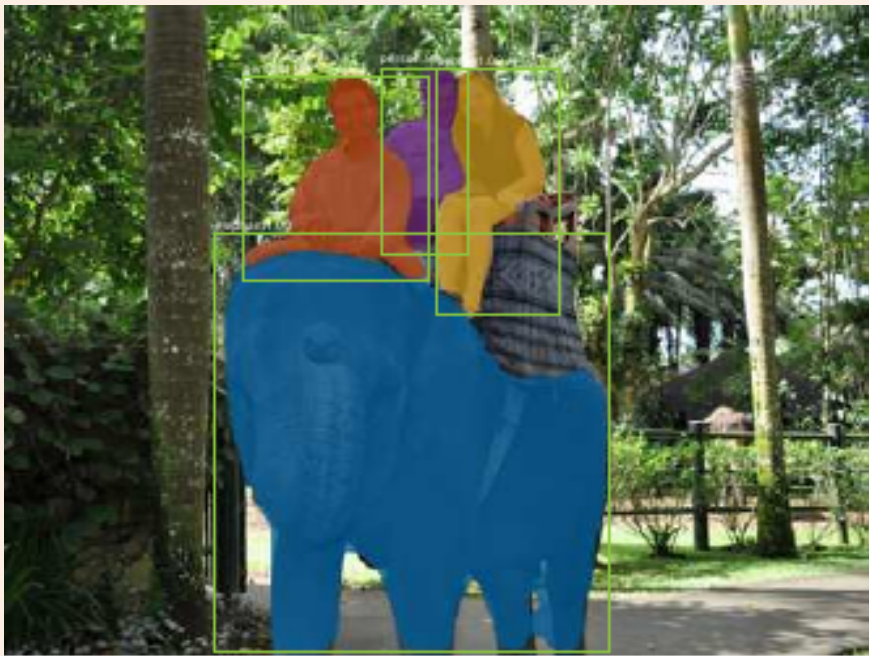
Compare scattered visibility, one missing block, and a regular grid.

The same fraction of hidden pixels can leave very different spatial evidence.

Downstream Tasks Reuse the Pretrained Encoder

Downstream task = the task we want to solve after pretraining.

Image → pretrained encoder → task head → task prediction



Boxes locate objects; colored masks outline them.
Task illustration: Mask R-CNN, not an MAE output.

Task	What does the head predict?
Classification	An image category
Object detection	Object locations and categories
Segmentation	Labels or masks for image pixels

Frozen encoder

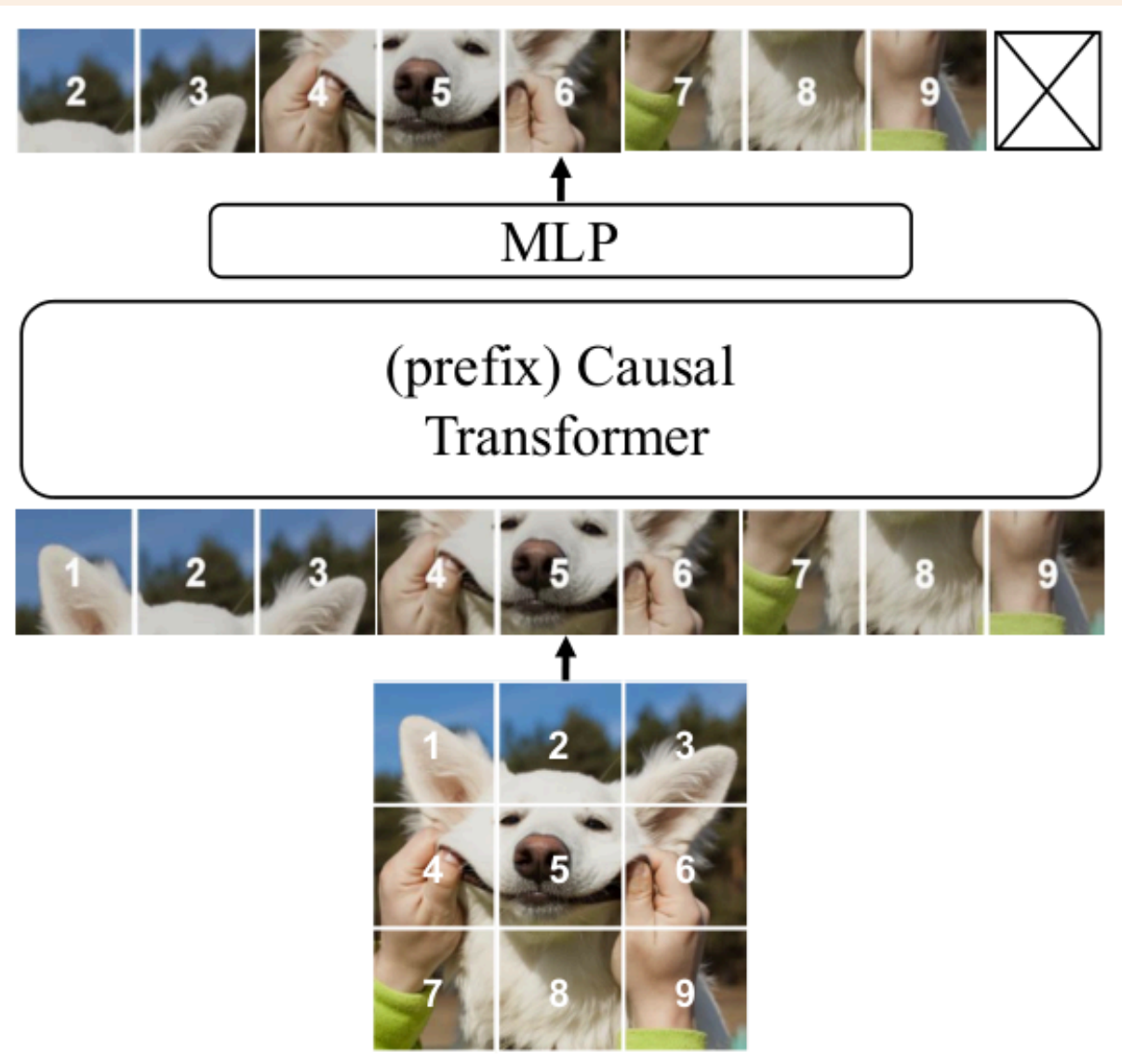
Train only the head: linear / attention probing.

Fine-tuning

Train the task head and update the encoder.

Replace the reconstruction decoder with the task head; evaluate the task you need.

AIM Predicts the Next Patch in an Image Scan



Original paper · Figure 2

Read the figure from bottom to top

Input: split the image into patches and choose an order: 1, 2, 3, ...

Target: shift by one position.
The feature at patch i predicts patch $i + 1$.

Prediction head: the MLP outputs the next patch's pixel values.

$$\hat{x}_{i+1} = H_{\theta}(e_i)$$

The scored target must stay hidden from its predicting position.

MAE hides a subset of patches. AIM predicts later patches from earlier context.

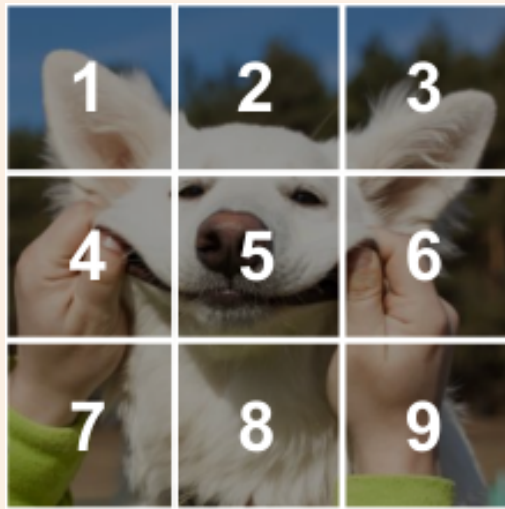
AIM Hides Each Scored Target from Its Predicting Position

Output row sees →	x ₁	x ₂	x ₃	x ₄	x ₅	Next target
Input x ₁	●	●	—	—	—	x ₂ · excluded
Input x ₂	●	●	—	—	—	x ₃ · scored
Input x ₃	●	●	●	—	—	x ₄ · scored
Input x ₄	●	●	●	●	—	x ₅ · scored
Input x ₅	●	●	●	●	●	end

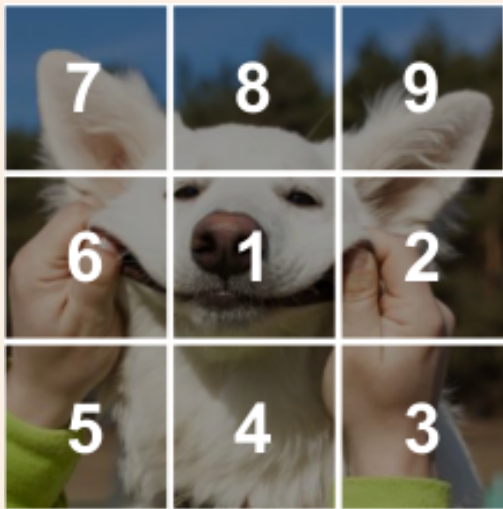
Prefix S = 2: x₁ and x₂ see each other. Later rows see only their prefix and past/current inputs.

The x₂ output predicts x₃ without seeing x₃. Prefix targets are excluded.

An Image Scan Order Is a Choice Made for Training



Raster



Spiral



Checkerboard

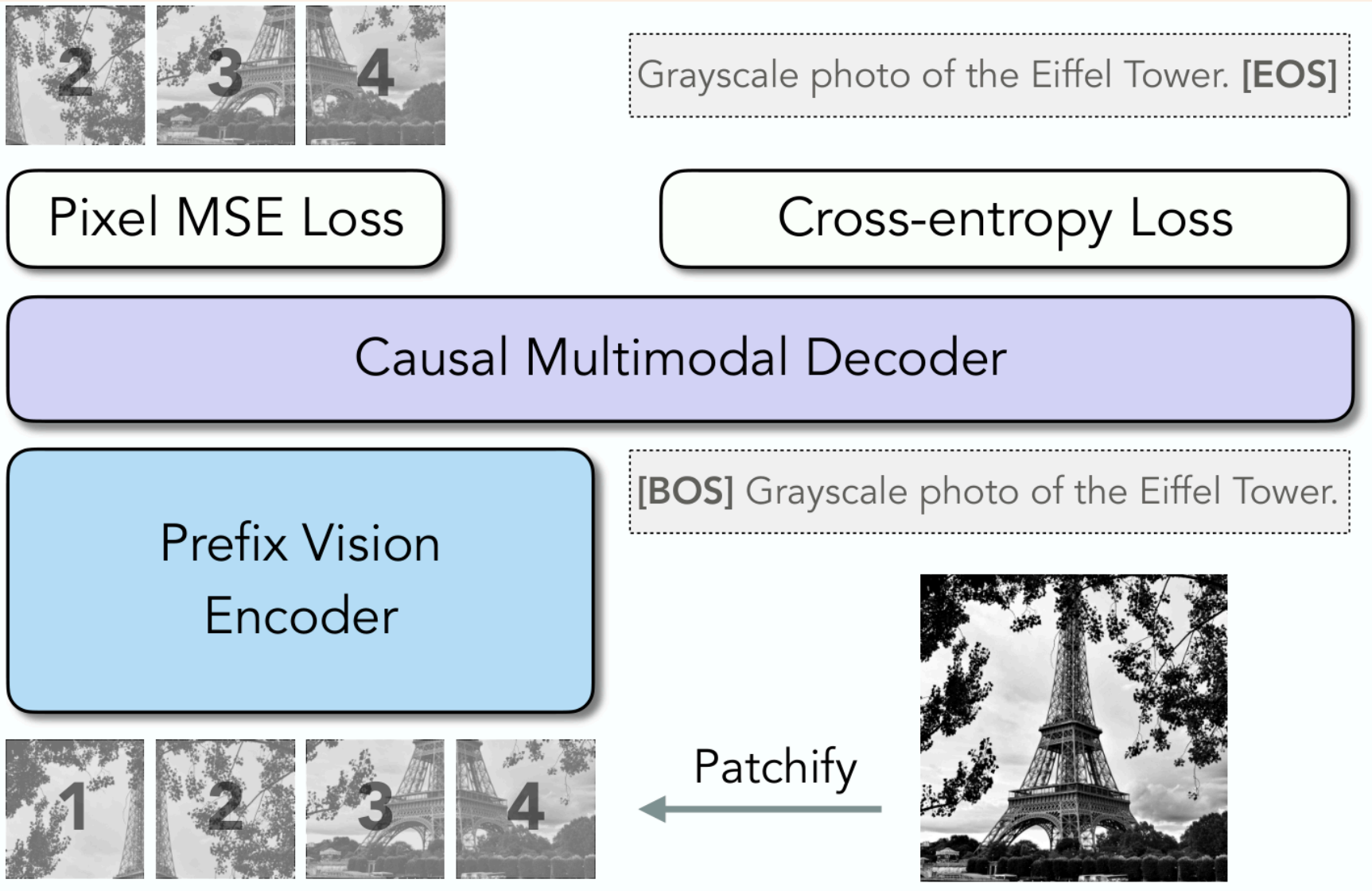


Fixed random

For each pattern, identify which patch comes before patch 5.

Autoregressive order in an image is not physical time.

AIMv2 Learns from Image Patches and Paired Text



Original paper · Figure 1

Two targets, one training sequence

Image positions:
predict the next patch's pixels.
Loss: pixel MSE.

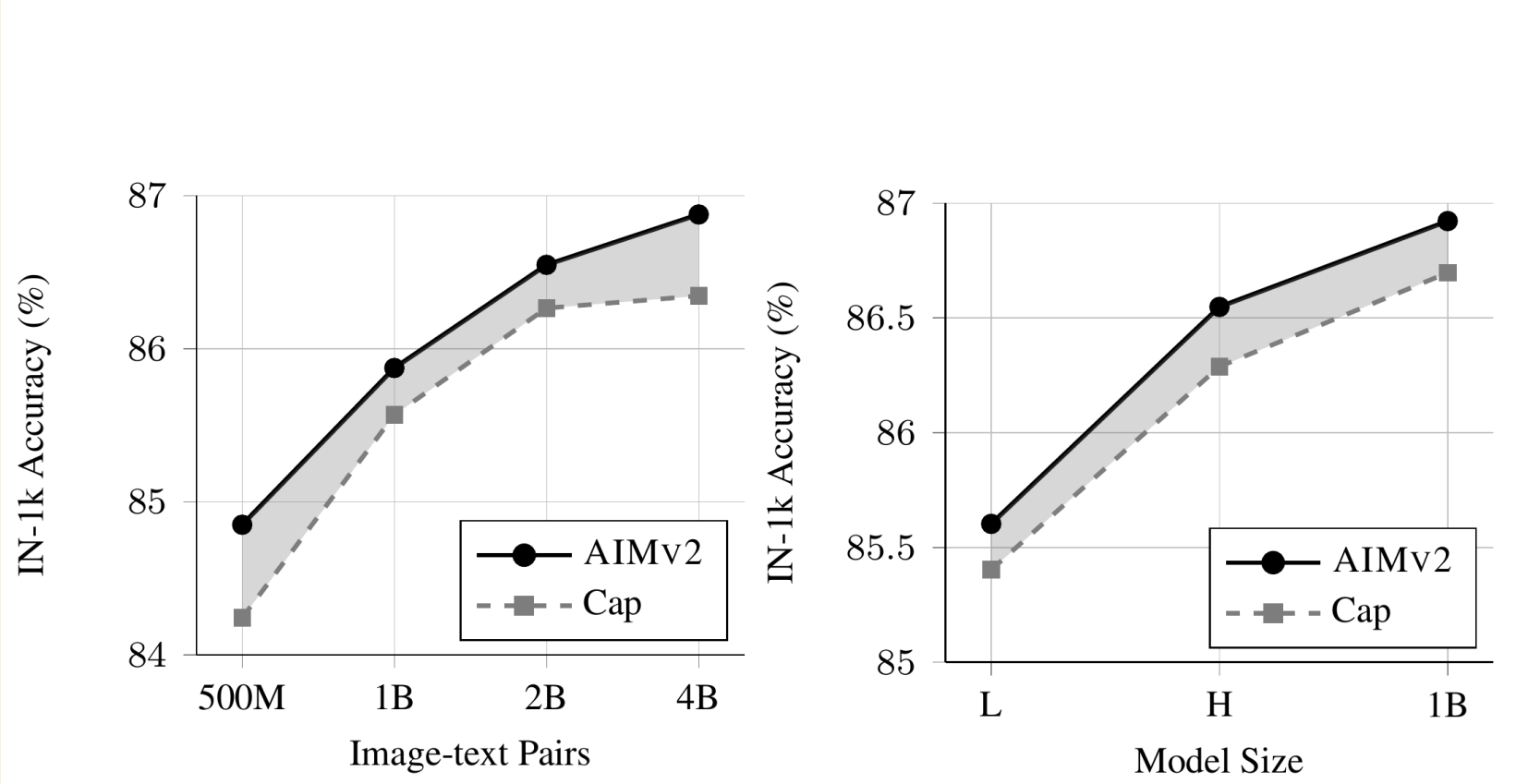
Text positions:
predict the next caption token.
Loss: cross-entropy.

$$\mathcal{L} = \mathcal{L}_{\text{text}} + \alpha \mathcal{L}_{\text{img}}$$

The image comes first; text prediction can use its visual features.

Image and text prediction jointly train the vision encoder used downstream.

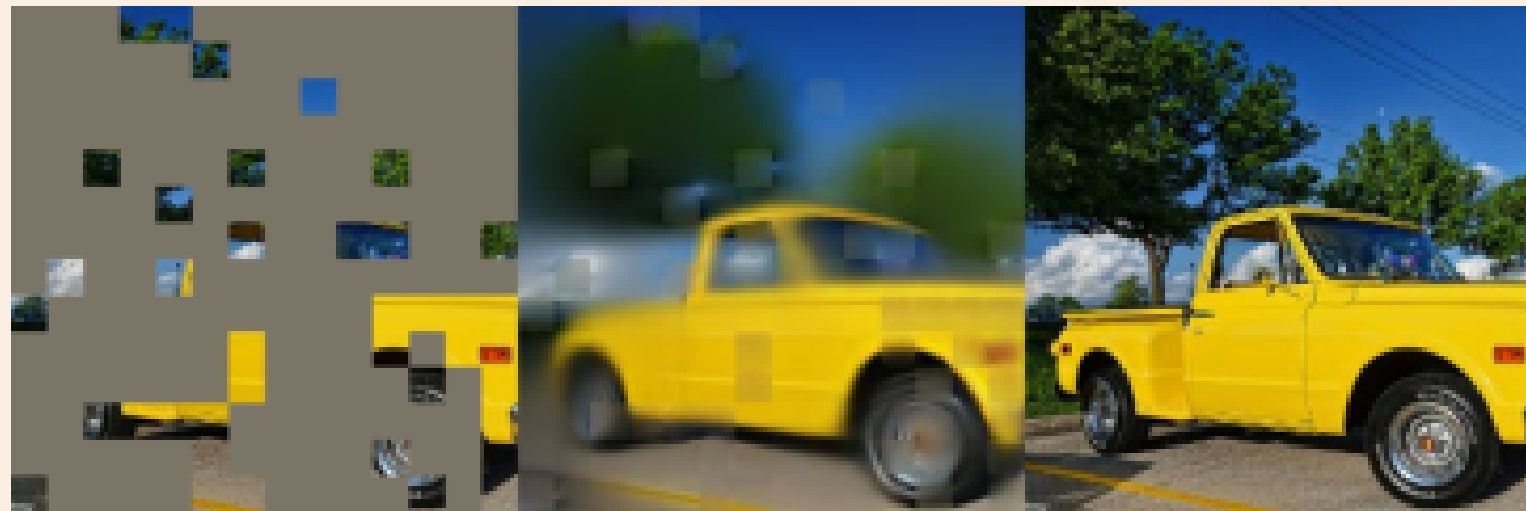
AIMv2 Tests What Image Prediction Adds to Captioning



AIMv2: image + text objectives. Cap: caption-only baseline. Attentive probe: freeze encoder, train attention pooling + classifier.

Compare objectives within a controlled setup before comparing headline scores.

Does Reconstruction Learn the Features We Need?



Masked input

Reconstruction

Original

The training target is missing pixels.

The downstream target might be an object's category or location.

Original MAE example; reconstruction alone does not evaluate its features.

Which details deserve model capacity?

Pixel loss scores texture and lighting as well as shape.

Is there only one possible answer?

Several completions may fit the same visible evidence.

Are the features easy to reuse?

A decoder can use information that a small task head may struggle to read.

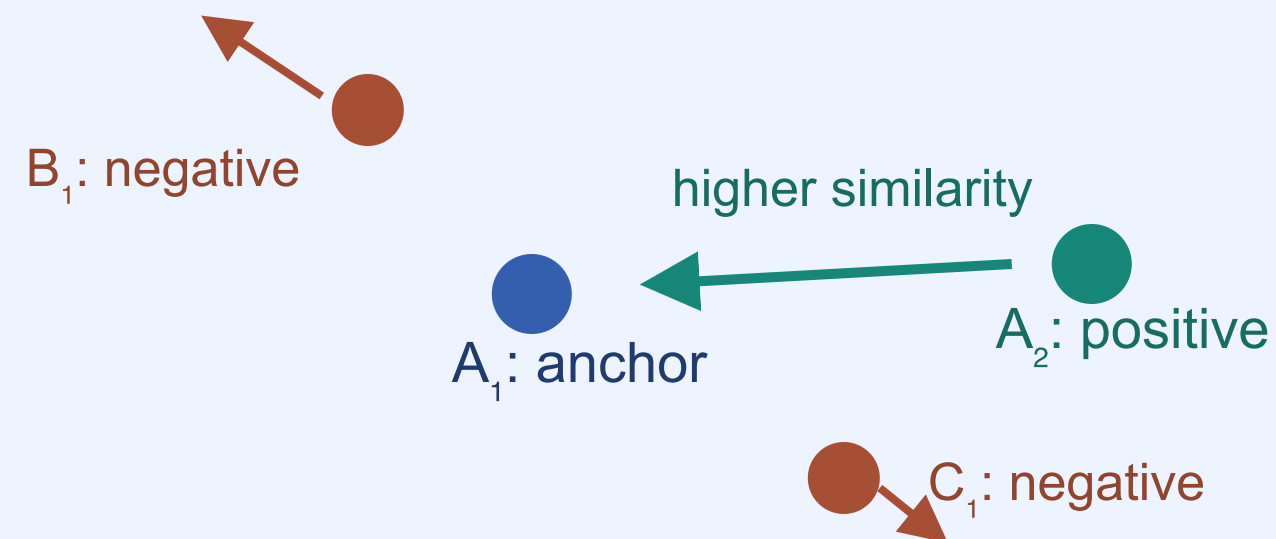
Another training signal: which views should have similar features?

Reconstruction can learn strong features; its objective does not directly specify every downstream need.

Contrastive Learning Identifies Which Views Belong Together

Create different views, then learn features that identify the matching view among candidates.

Embeddings of different views



A_1 and A_2 come from the same image A .

B_1 and C_1 come from other images. Positions are schematic.

1. Construct the pairing

Sample two views of one example.

2. Encode and compare

Map each view to an embedding; compute similarity.

3. Favor the positive over negatives

Train the matched view to receive the higher score.

$$s(A_1, A_2) > s(A_1, B_1), s(A_1, C_1)$$

The target is which views match. The loss compares embeddings, not reconstructed pixels.

Two Crops of One Image Supply a Positive Pair



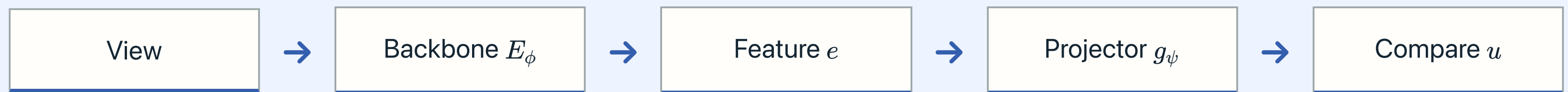
View A_1



View A_2

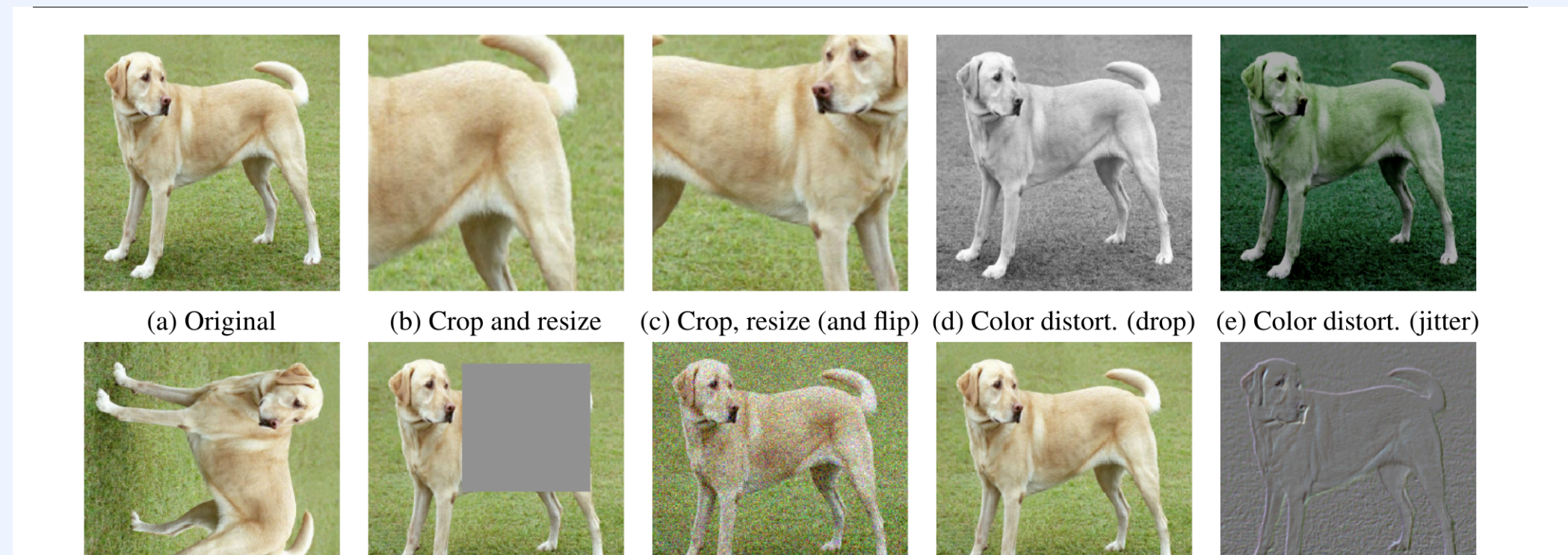
Both came from image A.

Other images supply candidate mismatches.



The loss compares u ; downstream readouts commonly use e .

Augmentations Change the Evidence Shared by Two Views



SimCLR's main recipe combines crop/resize, color distortion, and blur.

Which transformation would harm an orientation or color-sensitive task?

The figure surveys candidates; the main recipe does not use every pictured transform.

Why Does SimCLR Change Each Crop's Colors?

Find the crop from the **same source image** as the anchor.

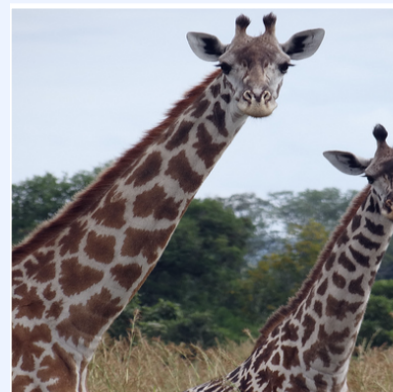
Only crop the images



Anchor



Candidate 1



Candidate 2

"Pick the blue candidate" → Candidate 1.

The rule gets this pair right without recognizing the tower.

Change each crop's colors separately



Anchor



Candidate 1



Candidate 2

"Pick the blue candidate" → Candidate 2.

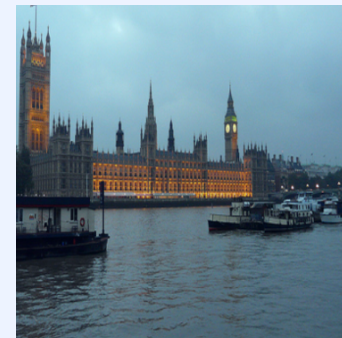
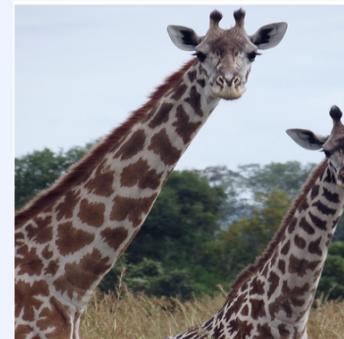
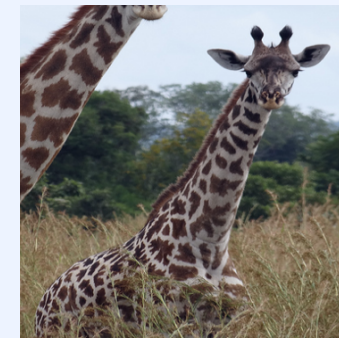
Now the same color rule selects the wrong image.

Candidate 1 is still the positive. Its source image did not change.

Independent color changes make color alone less reliable for matching views.

Illustrative crops and color changes; no trained model or measured scores are shown.

Three Source Images Produce Six Views

 A_1  A_2  B_1  B_2  C_1  C_2

Anchor A_1 chooses among five candidates: A_2 , B_1 , B_2 , C_1 , C_2 .

A_2 is positive. A_1 itself is excluded from the candidate set.

Each Row Is a Five-Candidate Classification Problem

	A ₁	A ₂	B ₁	B ₂	C ₁	C ₂
A ₁	×	+	−	−	−	−
A ₂	+	×	−	−	−	−
B ₁	−	−	×	+	−	−
B ₂	−	−	+	×	−	−
C ₁	−	−	−	−	×	+
C ₂	−	−	−	−	+	×

× · self-pair

Excluded from the denominator.

+

One per row, from the same source image.

− · negative

Four competing candidates in this toy batch.

Both $A_1 \rightarrow A_2$ and $A_2 \rightarrow A_1$ are training terms.

Similarities Become a Probability for the Paired View



$$\ell_{i,j} = -\log p_{ij} = -\log \frac{\exp(s_{ij}/\tau)}{\sum_{k \neq i} \exp(s_{ik}/\tau)}$$

$s_{ik} = \cos(u_i, u_k)$; $\tau > 0$ sets the scale.

Sum over every candidate, including the paired view j .

This is a classification task whose correct candidate is supplied by view sampling.

How the Contrastive Loss Trains the Representation

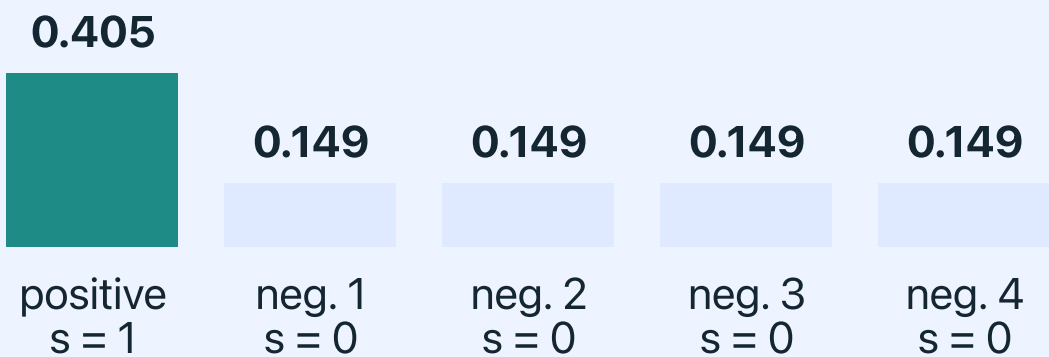
Image → encoder → **feature e** → projector → **vector u** → contrastive loss

Temperature changes the probabilities

Temperature τ 1.0

Clear positive

Similar negative



Positive probability = 0.405 · anchor loss = 0.905

The loss changes the similarities

$$\frac{\partial \ell}{\partial s_k} = \frac{p_k - \mathbf{1}[k = +]}{\tau}$$

Positive: gradient descent pushes its similarity up.

Negatives: higher-probability negatives receive stronger pressure to move down.

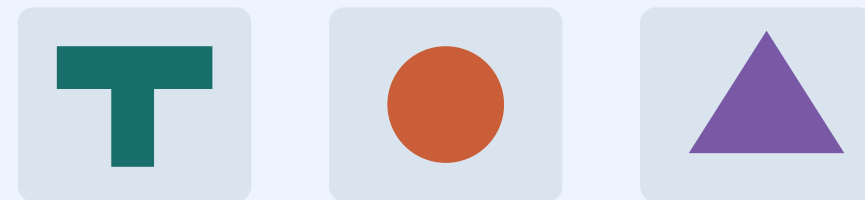
These gradients pass through the projector and encoder.

Lower τ sharpens the current preference. It can favor a negative if that negative has the highest score.

The loss compares u; downstream tasks usually reuse e. Both are learned during pretraining.

Why Does Contrastive Learning Need Negatives?

Suppose we only ask two views of each image to agree.



Different observations



The same encoder



C, C, C, ...

One vector for every input

Only pull positive pairs together

$$\ell_{\text{pair}} = \|u^{(1)} - u^{(2)}\|^2 = 0$$

The constant solution gets a perfect loss.

It distinguishes nothing.

Ask the positive to beat four negatives

$$p_+ = \frac{1}{5}, \quad \ell = -\log p_+ = \log 5$$

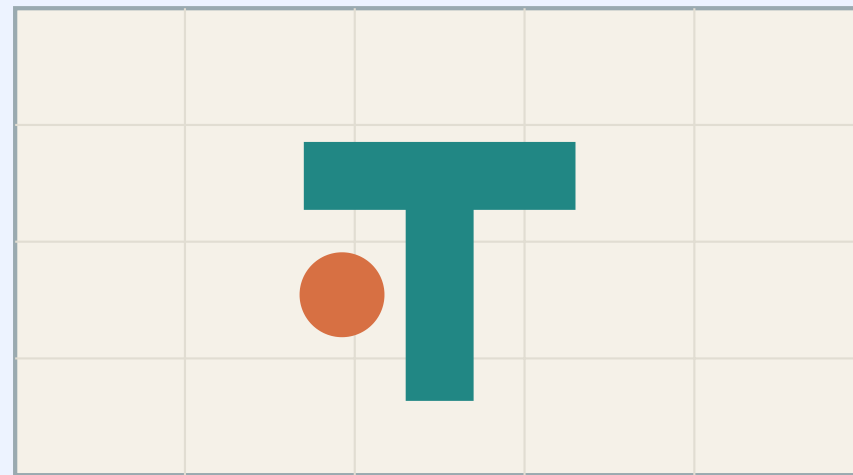
All candidates tied is a poor match.

Better separation can lower the loss.

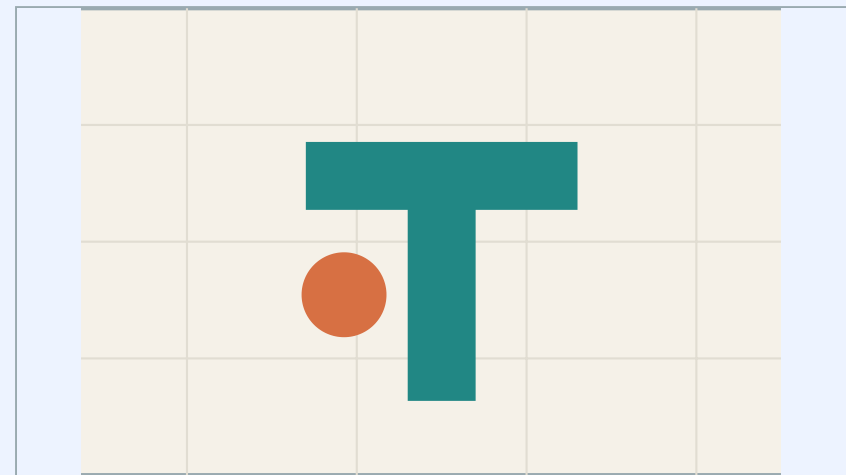
Can we learn without negatives? Yes, with another way to prevent collapse. Simply deleting negatives is insufficient.

A Different Image Can Be a Bad Negative

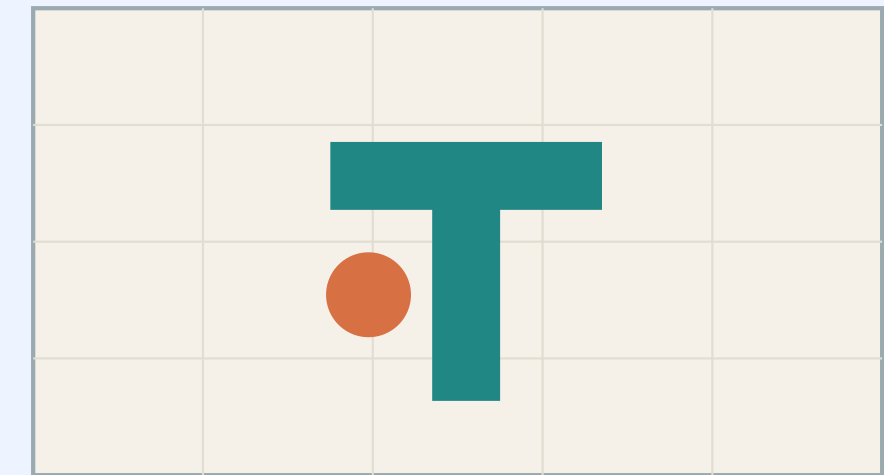
Imagine that a batch contains two nearby frames from the same robot video.



Anchor: frame A



Positive: another crop of A



Negative: frame B, a moment later

Training rule: match A with its crop.
Lower A's similarity to B because B has a different image ID.

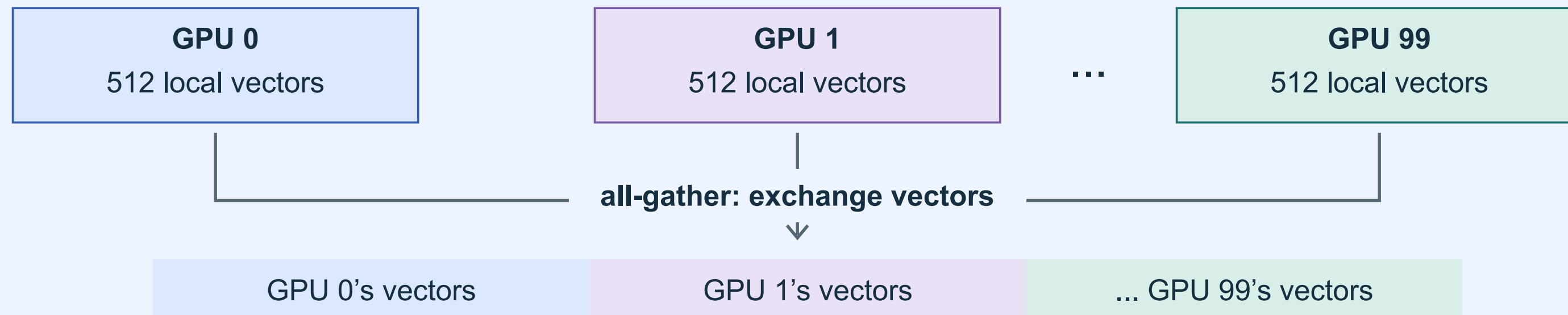
What we wanted: similar object configurations should remain easy to recognize as similar.

A false negative is a pair that the sampler separates even though it shares information we want to preserve.

More negatives add comparisons, but can also add false negatives, score computation, and communication.

How Do 100 GPUs Share Negative Samples?

$100 \text{ GPUs} \times 256 \text{ images/GPU} \times 2 \text{ views/image} = 51,200 \text{ vectors.}$



Each GPU can now compare against all 51,200 vectors.

1. Score local anchors.

Each GPU computes $512 \times 51,200$ similarities.

Other GPUs compute their own anchor rows.

2. Identify the matching view.

Exclude the anchor itself. Keep 1 positive.

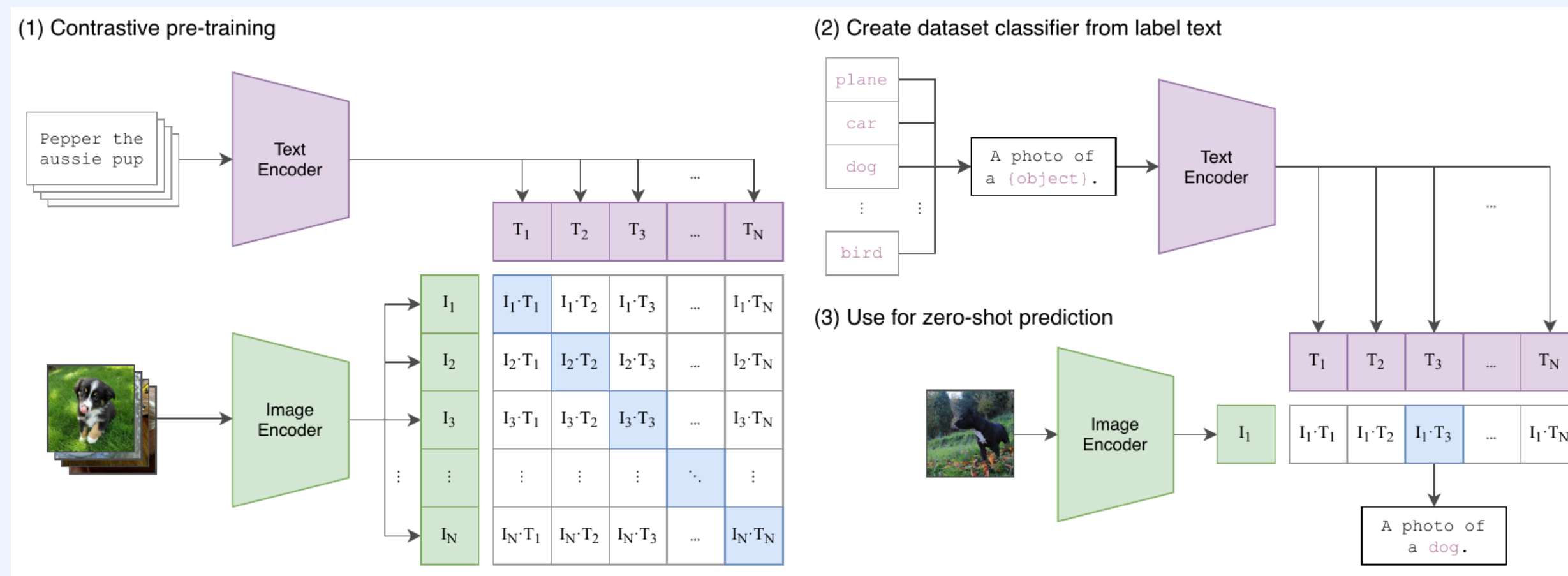
The other **51,198 candidates are negatives.**

3. Backpropagate and synchronize. Return gradients through gathered vectors, then average the model gradients across GPUs.

Negatives come from the other examples already in the batch. The exchange carries feature vectors.

CLIP Matches Images with Their Captions

The positive is the observed image–caption pair. Other captions in the batch act as negatives.



During training

Two encoders form an image $\times$ text similarity matrix.
Row and column softmax select the paired examples.

After training

Compare an image with class prompts such as
"a photo of a dog" for zero-shot classification.

Text supplies semantic supervision at scale. Captions may omit the geometry or motion a world model needs.

SigLIP Scores Each Image–Text Pair Independently

CLIP: choose the paired caption

A softmax compares one caption with all competing captions in its row.



One image, two separate decisions

SigLIP: is this pair a match?

$$\ell_{ij} = -\log \sigma(y_{ij}(a s_{ij} + b))$$

y = +1 for observed pairs, −1 otherwise.
s = cosine similarity; a > 0 and b are learned.

Caption paired with this image	Label	Loss encourages
"A tower beside a river" Observed caption	+1	A higher matching score
"A pair of giraffes" Unpaired caption	−1	A lower matching score

No batch-wide softmax denominator. Pair losses can be computed and accumulated block by block.

SigLIP still uses negative pairs. Cross-device negatives still require communication, and false negatives remain.

So Far: What Do These Training Targets Miss?

	Reconstruction / information recovery	Contrastive learning
The training answer	Original content or structure DAE · MAE · AIM / AIMv2	Which examples should match? SimCLR · CLIP · SigLIP
What can go wrong?	Easy clues can solve the task. Pixel targets can reward irrelevant detail or have multiple plausible answers.	Positive views can erase useful differences. False negatives can separate similar cases. Large pools cost computation and communication.
The world-model test	Does recovering the target expose position, contact, and motion?	Do the pair labels preserve the differences that later predictions need?

Thursday: Self-Distillation & Prediction in Feature Space

DINO: match teacher outputs. **data2vec:** predict contextual features.
JEPA family: predict features of hidden image regions or video.

If the targets are learned too, how do we keep them informative without negatives?