14 Object Centered Methods

Master of Science in Engineering
FS 2025

Reader

MSE im Modul FTP MachLe (WUCH)

Object-Centric Methods: Detailed Overview

This report gives an overview of object-centric methods, including their core ideas, architectures, training/inference procedures, empirical findings, and limitations.

Inhaltsverzeichnis

1 Object-Oriented Prediction (OP3)

Core Idea & Motivation: OP3 (Object-centric Perception, Prediction, and Planning) is a latent-variable framework for model-based reinforcement learning that learns to decompose raw images into a set of "entity" (object) representations without any direct supervision. By enforcing a factorization of the latent state into object-level variables, OP3 aims to generalize to scenes with different numbers or configurations of objects than those seen during training. OP3 uses these learned object representations to predict future frames and plan actions in a block-stacking environment, demonstrating superior generalization compared to both supervised object-aware baselines and pixel-level video predictors [1, 2].
Model Architecture:

(a) **Latent Factorization:** OP3 models the hidden state $\mathbf{s}_t$ at time t as a collection of K object-level latent vectors $\{\mathbf{z}_t^{(i)}\}_{i=1}^{K}$. Each $\mathbf{z}_t^{(i)}$ is intended to represent one object (or entity) in the scene. All K slots share the same neural transition and decoder functions, ensuring *per-object parameter sharing (entity abstraction)* [1, 2].

(b) **Interactive Inference (Slot Binding):**
- The primary challenge is *grounding* these abstract latent variables to actual image regions (object instances). OP3 treats this as a *state-factorized*

partially observable Markov decision process (POMDP).

- It uses an *amortized inference network* that takes in the current image and the previous beliefs to iteratively refine each slot's latent via a learned inference step. During each iteration, the network (a small convolutional encoder + object-specific attention) "proposes" which part of the image belongs to each latent slot, allowing the model to bind $\mathbf{z}_t^{(i)}$ to a physical object. This procedure is run for multiple steps per frame to resolve occlusions and maintain consistent object identities over time [3, 2].

(c) **Transition & Decoder:**

- *Transition Model:* Each slot latent $\mathbf{z}_t^{(i)}$ is updated to $\mathbf{z}_{t+1}^{(i)}$ by passing it (and optionally aggregated information from the other slots) through a shared, locally scoped transition network (e.g., an MLP or RNN). Because the same transition is applied to all slots, the model is *permutation-invariant* with respect to object ordering.
- *Decoder (Reconstruction):* Given $\{\mathbf{z}_t^{(i)}\}$, OP3 reconstructs the next image $\hat{\mathbf{x}}_{t+1}$ by first decoding each slot into an "object image" and a mask via a small CNN decoder, then compositing these K object renditions via an alpha-blending scheme (along with a learned background latent).

(d) **Planning & Control:**

- Once trained, OP3 uses the object-level transition model to *roll out* future latent states under candidate actions, allowing planning via, for example, model predictive control (MPC). In their evaluations, OP3 uses the learned dynamics within an MPC loop to choose actions that achieve block-stacking goals.

Training Objective:

- OP3 is trained end-to-end with a combination of:
  (a) *Reconstruction Loss:* Encouraging the decoded composite image $\hat{\mathbf{x}}_{t+1}$ to match the true next frame $\mathbf{x}_{t+1}$.
  (b) *KL Regularization:* A KL divergence between each slot's posterior $q(\mathbf{z}_t^{(i)} \mid \mathbf{x}_{1:t})$ and a prior $p(\mathbf{z}_t^{(i)})$, to prevent degenerate solutions.

(c) *Planning Loss (Optional):* For certain tasks, an auxiliary loss may encourage the transition to align with observed object motions, but OP3's primary planning occurs at test time in an MPC loop rather than via a separate loss term.

Empirical Findings:

- *Generalization to Novel Configurations:* On a suite of 2D block-stacking tasks, OP3—trained with up to 3 blocks—could *generalize zero-shot* to scenes with up to 5 blocks in new spatial arrangements, outperforming both a fully supervised object oracle (which had access to object masks) and a pixel-level video predictor (which did not factorize by object) by 2–3$\times$ in prediction accuracy [1, 2].
- *Planning Performance:* When used for MPC on stacking tasks, OP3 achieved higher success rates (e.g., stacking 4 blocks) than baselines that lacked explicit object factorization.

Limitations & Subsequent Work:

- *Inference Complexity:* The interactive inference routine (iterating multiple rounds of binding) can be computationally expensive, particularly as scenes or the number of slots K grow.
- *Fixed Number of Slots:* OP3 requires specifying the maximum number of objects K in advance, which may not match every scene exactly, potentially leading to empty slots or slot "over-segmentation."
- *Assumption of Non-Overlapping Objects:* In highly cluttered or heavily overlapping scenes, OP3's binding procedure may struggle to assign distinct latents to distinct objects.
- Later works (e.g., video-extension models and adaptive slot methods) build on OP3's core idea of object factorization to improve scalability and handle variable object counts [3].

2 Slot Attention

Core Idea & Motivation: Slot Attention is a *differentiable module* that bridges a set of unstructured per-pixel (or patch) features and a fixed number of "slots," each intended to capture one object or component in the input. Unlike prior attention-based or capsule-based methods that either specialized slots to certain object types or required ad hoc supervision, Slot Attention learns to assign each slot to *any* object in the scene in an *unsupervised* or *weakly supervised* manner. Through an iterative attention-based update, slots compete for feature support, leading to a permutation-invariant, object-centric latent set that can be used for downstream tasks (e.g., image decomposition, property regression) [4]. The original code from google can be found here.

Model Architecture:

(a) **Input Features:** Let the input image be processed by a convolutional encoder (e.g., a CNN) to produce a feature map $\mathbf{F} \in \mathbb{R}^{H \times W \times D}$. We flatten it into a set of $N = H \times W$ vectors $\{\mathbf{f}_n\}_{n=1}^N$, each $\mathbf{f}_n \in \mathbb{R}^D$.

(b) **Initial Slots:** Initialize K slot vectors $\{\mathbf{s}_k^{(0)}\}_{k=1}^K$ by sampling from a learned Gaussian prior (or simply from parameters), so that each slot $\mathbf{s}_k^{(0)} \in \mathbb{R}^D$.

(c) **Iterative Attention Updates:** For T rounds (e.g., $T = 3$):

   a) *Compute Attention Scores:*

$$a_{nk}^{(t)} = \mathbf{f}_n^\top \mathbf{W}_q \, \mathbf{s}_k^{(t-1)}, \quad \alpha_{nk}^{(t)} = \frac{\exp\left(a_{nk}^{(t)}\right)}{\sum_{k'} \exp\left(a_{nk'}^{(t)}\right)} \quad \forall n, k.$$

   Here, $\mathbf{W}_q$ is a learnable projection, and $\alpha_{nk}^{(t)}$ is the attention weight of slot k for feature n. Because we normalize across slots k for each pixel n, the slots *compete* to explain each feature.

   b) *Aggregate Slot Inputs:*

$$\hat{\mathbf{s}}_k^{(t)} = \sum_{n=1}^{N} \alpha_{nk}^{(t)} \mathbf{W}_v \, \mathbf{f}_n,$$

   where $\mathbf{W}_v$ projects features into a "value" space.

c) *Slot Update (GRU-like):*

$$\mathbf{s}_k^{(t)} = \mathrm{GRU}\Big(\hat{\mathbf{s}}_k^{(t)},\, \mathbf{s}_k^{(t-1)}\Big),$$

or using a residual MLP:

$$\mathbf{s}_k^{(t)} = \mathbf{s}_k^{(t-1)} + \mathrm{MLP}\Big(\hat{\mathbf{s}}_k^{(t)}\Big).$$

These updates allow the slot to refine its latent representation by focusing on its assigned features.

After T rounds, the final slots $\{\mathbf{s}_k^{(T)}\}_{k=1}^K$ serve as the *object-centric embeddings*.

(d) **Decoder / Downstream Use:**

- *Unsupervised Object Discovery:* Each slot's latent $\mathbf{s}_k^{(T)}$ is concatenated with a learnable positional embedding and passed through a small decoder (deconvolutional or MLP) to predict an *attention mask* $m_k(x, y)$ and a *pixel reconstruction* for that slot. The full image is reconstructed by compositing these K "slot images" via soft-mask blending.
- *Supervised Set-Prediction:* The slots $\{\mathbf{s}_k^{(T)}\}$ are fed to an MLP trained to predict a *set* of target properties (e.g., object class, color). Because slots are order-invariant, one can match the predicted slot set to the ground-truth set via Hungarian matching during training.

Training Objectives:

- *Unsupervised Image Decomposition (Object Discovery):*
  (a) *Reconstruction Loss:* Encourage the composite of slot decodings to match the input image.
  (b) *KL Regularization (optional):* If using a VAE variant on each slot, include a KL penalty on each slot's latent.

  The model discovers a segmentation of the image into K masks $\{m_k\}$ (soft masks summing to 1 per pixel) and a reconstructed pixel value per slot.
- *Supervised Property Prediction:*

(a) *Slot–Property Matching:* Use a bipartite matching between slots and labeled objects; then minimize the error in predicting each object's properties (e.g., color, shape).

(b) *Permutational Invariance:* Because slot order is arbitrary, losses are computed on matched pairs to avoid imposing an order.

Empirical Findings:

- *Unsupervised Object Discovery:* On datasets like CLEVR6 (images with up to 6 objects), Slot Attention with $K = 7$ slots successfully decomposed scenes into object masks that align well with ground-truth segmentations, despite no mask supervision [4].
- *Generalization to Unseen Compositions:* When trained on scenes of up to 5 objects, Slot Attention still segmented scenes with 6 objects at test time, demonstrating systematic generalization [4].
- *Set Property Prediction:* On tasks requiring predicting object attributes (e.g., color, shape) as a set given an image, Slot Attention outperformed prior methods (like IODINE) in accuracy and training speed [4].

Limitations & Extensions:

- *Fixed Slot Count K:* The original Slot Attention requires specifying a fixed K. If the scene has fewer objects, some slots become "empty"; if more, the model may split an object across multiple slots. Subsequent work (e.g., Adaptive Slot Attention, 2024) extends this to dynamically choose K per image.
- *No Explicit Object Identity Over Time:* Slot Attention as originally proposed is frame-by-frame; tracking the same slot across video frames requires additional mechanisms (e.g., ViMON).
- *Sensitivity to Input Feature Quality:* The quality of CNN-extracted features heavily influences performance; poor features can degrade slot assignments in complex scenes.

- *Computational Cost:* Iterative attention (e.g., 3 rounds per image) can increase latency, though typically still faster than iterative refinement methods like IODINE.
- *Ambiguity in Scenes with Similar Objects:* When multiple identical or highly similar objects appear, slots may "swap" their assignment across runs, which can affect downstream tasks that require consistent identity (though permutation invariance often mitigates this in set-based tasks).

3 GENESIS (and GENESIS-V2)

Core Idea & Motivation: GENESIS (Generative Scene Inference and Sampling) is an unsupervised, object-centric *generative model* that (1) decomposes a scene into a set of object (and background) latents, (2) captures relationships between those objects, and (3) can sample novel scenes by generative sampling of those latents. Unlike MONet or IODINE, which produce object masks sequentially but do not explicitly model *inter-object dependencies* in the generative process, GENESIS parameterizes a *spatial Gaussian mixture model (GMM)* over pixels, where each mixture component corresponds to an object. The component means (pixel-wise appearance) and mixture weights (mask) are decoded from a set of object latents that are either *inferred sequentially* or *sampled autoregressively* [5, 4].
Model Architecture (GENESIS):

(a) **Spatial GMM Formulation:** The generative model assumes each pixel $\mathbf{x}_i$ (at location i) is drawn from a *Gaussian mixture*:

$$p(\mathbf{x}_i \mid \{\mathbf{z}_k\}_{k=1}^{K}) \;=\; \sum_{k=1}^{K} \pi_{i,k}\,\mathcal{N}\Big(\mathbf{x}_i \mid \boldsymbol{\mu}_{i,k},\, \sigma^2\mathbf{I}\Big),$$

where K is the maximum number of "slots" (objects + background). Here:

- $\pi_{i,k} = \mathrm{softmax}_k\Big(s_{i,k}\Big)$ is the *pixel-wise mixture weight* (mask) for component k.
- $\boldsymbol{\mu}_{i,k}$ is the *reconstructed color* of component k at pixel i.
- Each component's parameters $\{\boldsymbol{\mu}_{i,k},\, s_{i,k}\}$ are functionally dependent on its latent $\mathbf{z}_k$.

One component (e.g., $k = K$) is reserved for the *background*; the other $K - 1$ represent foreground "objects."

(b) **Latent Inference (Encoder):** GENESIS uses an *RNN-based sequential inference* to extract object latents. Starting with the full image $\mathbf{x}$, at step $k = 1$ it infers $\mathbf{z}_1 \sim q(\mathbf{z}_1 \mid \mathbf{x})$. Then it reconstructs the first component's mask and appearance, subtracts their contribution from the image, and proceeds to infer $\mathbf{z}_2$ on the *residual*. In effect, this sequential "peeling" yields a set of latent vectors $\{\mathbf{z}_k\}_{k=1}^{K}$, each corresponding to one object (or the background).

- The inference network typically consists of: a shared CNN to extract convolutional features from the current residual image, an RNN (LSTM/GRU) to maintain context from previously inferred slots, and an MLP to produce $\mathbf{z}_k$'s posterior parameters (mean and variance).
- After inferring K slots, the final slot $\mathbf{z}_K$ is interpreted as the *background*.

(c) **Decoder (Generative Model):** Each latent $\mathbf{z}_k$ is passed through a small deconvolutional decoder that outputs for every pixel i:

- A *log-weight* $s_{i,k}$ (pre-softmax).
- A *mean color* $\boldsymbol{\mu}_{i,k}$.

The mixture weights $\{\pi_{i,1}, \dots, \pi_{i,K}\}$ are obtained via softmax over $\{s_{i,1}, \dots, s_{i,K}\}$ per pixel. Thus the final image likelihood is computed by plugging into the GMM formula above.

(d) **Autoregressive Prior (for Sampling):** To generate novel scenes, GENESIS defines a *chain rule prior* on $\{\mathbf{z}_k\}$:

$$p(\mathbf{z}_1)\, p(\mathbf{z}_2 \mid \mathbf{z}_1)\ \cdots\ p(\mathbf{z}_K \mid \mathbf{z}_{1:K-1}).$$

Each conditional prior $p(\mathbf{z}_k \mid \mathbf{z}_{1:k-1})$ is parameterized by an RNN (e.g., an LSTM) that consumes the previously sampled latents. This allows GENESIS to model *inter-object relationships*—for instance, capturing that the "table" should likely appear underneath the "cup," or that objects do not overlap in impossible ways. At generation time, one can sample $\mathbf{z}_1 \sim p(\mathbf{z}_1)$, then sequentially sample $\mathbf{z}_2 \sim p(\mathbf{z}_2 \mid \mathbf{z}_1)$, etc., and then decode each $\mathbf{z}_k$ to obtain a novel composition.

GENESIS-V2 Extensions: While GENESIS uses sequential RNN inference and decoding, *GENESIS-V2* (2021) introduces a *differentiable clustering* approach (IC-SBP: "Identifiable Clustering via Stick-Breaking Process") that:

(a) *Removes the need for a fixed ordering* in inference (no RNN).

(b) *Automatically infers a variable number of slots* by sampling cluster counts via a stick-breaking prior.

(c) *Scales better to real-world images* (e.g., COCO) by avoiding per-slot RNN iterations.

GENESIS-V2 encodes all pixels into embeddings, then performs *differentiable clustering* to group them into up to K clusters (with a learned upper bound but not forced to use all K). Each cluster yields a latent $\mathbf{z}_k$. These cluster latents are then decoded similarly to GENESIS. Empirically, GENESIS-V2 outperforms both GENESIS and MONet on unsupervised segmentation metrics and FID scores for generation on real-world datasets, showing improved object grouping and realistic scene synthesis [6, 4].
Training Objective: The objective is a standard *ELBO* for latent-variable models:

$$\mathcal{L} = \underbrace{\mathbb{E}_{q(\mathbf{z}_{1:K}|\mathbf{x})}\Big[\log p(\mathbf{x} \mid \mathbf{z}_{1:K})\Big]}_{\text{Reconstruction (GMM) term}} - \underbrace{\mathrm{KL}\Big(q(\mathbf{z}_{1:K} \mid \mathbf{x}) \,\|\, p(\mathbf{z}_{1:K})\Big)}_{\text{Latent prior term}}.$$

For GENESIS, q is the sequential inference network and $p(\mathbf{z}_{1:K})$ is the autoregressive prior. For GENESIS-V2, q is the differentiable clustering posterior and p is a stick-breaking prior.
Empirical Findings:

- On *CLEVR-derived* datasets (synthetic scenes of simple 3D objects), GENESIS matched or outperformed MONet and IODINE in:
  (a) *Scene Decomposition Metrics:* segmentation IoU and ARI (Adjusted Rand Index) for grouping pixels into object masks.
  (b) *Scene Generation Quality:* FID scores (with lower being better) for newly sampled images. GENESIS's autoregressive prior produced more coherent "joints" (relative object placements) than MONet, which lacks an explicit inter-object model [5, 4].

- On *real-world datasets* (e.g., Sketchy, APC), GENESIS-V2 showed improved unsupervised segmentation and generative sample quality over GENESIS and MONet, demonstrating that learned clustering and variable slot usage generalize beyond synthetic simple scenes [6, 4].

Limitations & Later Directions:

- *Fixed Slot Capacity in GENESIS:* The original GENESIS requires specifying an upper bound K, and infers exactly K latents (with one slot forced to be background). This can waste capacity when fewer objects are present, or fail when more than K objects appear. GENESIS-V2's clustering alleviates this by allowing variable slot counts.
- *Computational Overhead:* Sequential RNN inference and decoding in GENESIS scales poorly to large images and high K. GENESIS-V2 improved scalability, but real-time use in robotics or video still remains challenging.
- *Lack of Temporal Dynamics:* Both GENESIS and GENESIS-V2 operate on static images; they do not model object dynamics across frames. Extending to video (e.g., coupling with a learned transition model) is nontrivial.
- *Handling Occlusions & Depth:* While GENESIS decomposes layers via mixture masks, it does not explicitly reason about occlusion order or 3D depth; this can lead to artifacts when objects heavily overlap or have complex lighting.
- *Evaluation on Complex Scenes:* Later work explores combining GENESIS-style object latents with spatial-temporal models (e.g., dynamic NeRFs) to handle video and 3D reconstruction.

4 MONet (Multi-Object Network)

Core Idea & Motivation: MONet is one of the first unsupervised architectures to *jointly learn* to (a) segment an image into object masks and (b) represent each object with a latent vector, all in an *end-to-end VAE framework*. Its key insight is to combine a *recurrent attention network* (that produces soft masks for successive objects) with a *VAE* that encodes each masked region. Unlike prior methods (e.g., AIR) that sequentially inferred object latents but could not reconstruct them well, MONet's combination of attention and VAEs allows accurate object-level reconstruction and yields interpretable object representations [7, 4].

Model Architecture:

(a) **Recurrent Attention Network (Mask Proposal):**

- Given an input image $\mathbf{x}$, MONet first uses a shared *recurrent mask-proposal network* (an RNN over "slot" index k) that at iteration k takes as input:
  - The *current "reconstruction residual"* (i.e., $\mathbf{r}_{k-1} = \mathbf{x} - \sum_{j<k} \hat{\mathbf{x}}_j$, where $\hat{\mathbf{x}}_j$ are reconstructions of previously explained objects).
  - The previous hidden state of the mask RNN.
- It outputs a *soft mask* $m_k \in [0,1]^{H\times W}$ and a residual for the next step. Intuitively, m_1 attends to the most "salient" object; once that object is reconstructed, the attention is re-applied to the residual to find the next object.
- The final mask m_K (after $K-1$ objects) is taken to be the background mask $m_K = 1 - \sum_{j<K} m_j$.
- This produces K masks $\{m_1, \ldots, m_K\}$ that softly (i.e., with values in $[0,1]$) decompose the image into disjoint layers (with $\sum_k m_k(i) = 1$ at each pixel i).

(b) **Per-Slot VAE Encoding & Decoding:**

- For each mask m_k, define the *masked image patch* as $\mathbf{x}_k = m_k \odot \mathbf{x}$.
- An *encoder* CNN processes $\mathbf{x}_k$ (together with m_k concatenated) to produce a latent posterior $q(\mathbf{z}_k \mid \mathbf{x}_k, m_k)$. Each $\mathbf{z}_k \in \mathbb{R}^D$.
- A *decoder* network (also a small CNN or MLP) maps $\mathbf{z}_k$ to:
  a) *Reconstruction* pixels $\boldsymbol{\mu}_k(i)$ for each pixel i.
  b) Optionally, an updated mask logit $s_k(i)$ to refine segmentation (though in simple MONet, masks come only from the RNN).

- The overall per-slot reconstruction is $\hat{\mathbf{x}}_k(i) = \boldsymbol{\mu}_k(i)$, and the final full image is $\sum_{k=1}^{K} m_k(i)\,\hat{\mathbf{x}}_k(i)$.

(c) **Recurrent Loop Over Slots:**

- After reconstructing object k, MONet subtracts $m_k \odot \hat{\mathbf{x}}_k$ from the residual and proceeds to predict mask m_{k+1}.
- This continues for $K-1$ object slots; the residual at step K becomes the background input for the "background" VAE (slot K).

Training Objective: MONet optimizes a sum of *slot-wise VAE ELBOs*:

$$\mathcal{L} = \sum_{k=1}^{K} \Big\{ \underbrace{\mathbb{E}_{q(\mathbf{z}_k \mid \mathbf{x}_k, m_k)}\Big[\log p(\mathbf{x}_k \mid \mathbf{z}_k)\Big]}_{\text{Reconstruction}} - \underbrace{\mathrm{KL}\Big(q(\mathbf{z}_k \mid \mathbf{x}_k, m_k) \,\|\, p(\mathbf{z}_k)\Big)}_{\text{KL penalty}} \Big\}.$$

The reconstruction $p(\mathbf{x}_k \mid \mathbf{z}_k)$ is typically a Gaussian or Bernoulli over pixels masked by m_k. The masks m_k themselves are *not* explicitly optimized by MONet—rather, they are implicitly learned to maximize the total likelihood (slots that explain the image well are rewarded; others shrink).

Empirical Findings:

- On *3D synthetic datasets* (e.g., CLEVR, Shapes), MONet successfully decomposes scenes into object-aligned masks (e.g., separating front vs. back objects) in an unsupervised manner, yielding high segmentation IoU ($>$0.9 on simple scenes) and low reconstruction error [7, 4].
- When extended to *video (ViMON)* by adding a GRU per slot to carry over latent information from previous frames, MONet's framework tracks object identities over time, improving segmentation and tracking metrics compared to frame-by-frame MONet [3].

Limitations & Subsequent Improvements:

- *Fixed Slot Count K:* As in OP3 and Slot Attention, MONet requires a predetermined K. If K is too small, some objects are merged; if too large, extra slots learn degenerate "empty" representations.
- *Sequential Inference:* The RNN-based peeling can be slow, especially as K grows.

- *Mask Quality:* In complex real-world scenes, MONet's masks can be overly coarse or bleed between objects due to the simplicity of its recurrent attention.
- *Scaling to Real Images:* MONet struggles on high-resolution natural images without pretraining or large capacity; later work (e.g., GENESIS-V2) addresses this by using clustering instead of slot-by-slot RNN inference [6, 4].

5 CATER (Compositional Actions and TEmporal Reasoning)

Core Idea & Motivation: CATER is a *diagnostic video dataset* specifically designed to test a model's ability to perform *compositional action recognition* and *long-term temporal reasoning* in a fully *controllable, synthetic tabletop environment*. Unlike standard action recognition benchmarks (e.g., Kinetics) where spatial/scene biases can dominate, CATER's scenes are rendered with known 3D primitives (blocks, cones, rods) arranged on a table, ensuring that temporal cues (e.g., occlusion, containment, long-term tracking under occlusion) are essential to solve the tasks. CATER poses three tasks of increasing difficulty: (1) *Atomic Action Recognition*, (2) *Composite Action Recognition*, and (3) *Object Tracking*, each requiring the model to focus on different levels of spatio-temporal reasoning [8].
Dataset & Task Details:

(a) **Scene Generation:**
   - Scenes are *synthetically rendered* using a library of standard 3D primitives (cubes, spheres, cylinders).
   - Multiple objects (2–5) move on a tabletop under a controlled set of *atomic operations*:
     – *Rotate:* An object rotates in place.
     – *Contain/Release:* An object picks up another (e.g., a small sphere placed inside a cup), then later releases it.
     – *Slide:* An object slides along the table.
     – *Lift/Drop:* An object is lifted then dropped.
     – *Camera Motion:* Some versions include a slowly panning camera.

(b) **Tasks:**

- *Atomic Action Recognition (Task 1):* Given a short clip (2–4 seconds) that contains a single atomic operation (e.g., "move cube left"), classify which atomic operation occurred. This requires recognizing local object motion.
- *Composite Action Recognition (Task 2):* Each clip contains a *sequence of atomic operations* (e.g., "cube rotates, then sphere is contained, then both slide"). The model must label the clip with the full ordered sequence of atomic steps. This requires identifying *order* and *temporal segmentation*.
- *Object Tracking (Task 3):* Given a longer clip where objects may become occluded or contained, the model is asked a question like "Where is object X at time T?" (e.g., "Which object is on the table at the end?"). This demands long-term memory and reasoning about object permanence.

(c) **Diagnostics & Bias Control:** Because scenes and camera parameters are fully *observable and controllable*, one can systematically measure how performance degrades as (a) objects become more occluded, (b) clips get longer, or (c) camera motion increases. This design isolates whether a model truly leverages *temporal integration* vs. merely exploiting static appearance cues.

Leading Baseline Architectures & Findings:

(a) **3D CNNs (I3D, S3D, SlowFast, etc.):**

- On *Task 1*, 3D CNNs achieve high accuracy (≈98–99%) since recognizing a single atomic operation in a short clip mostly requires capturing short-term motion patterns.
- On *Task 2*, accuracy drops (≈60–70%), especially as the number of sequential steps grows beyond 3; 3D CNNs struggle to precisely order operations when they are composed of many quick steps [8].

(b) **Transformer-Based Models (TimeSformer, VideoSwin):**

- These models, with their *long-range attention*, improve composite action classification (reaching ≈75–80%), but still falter when fine-grained ordering or object-level details are needed, since many transformer patches may attend to irrelevant background pixels.

- *Temporal Resolution:* Downsampling to save computation (e.g., 8–16 fps) can hinder tracking tasks, where objects disappear behind occluders for multiple frames.

(c) **Object-Centric & Graph Models (Hopper, Loci):**

- *Hopper (2021)* proposes a multi-hop Transformer that explicitly reasons about *object tracks*:
  - First, track proposals for candidate objects are extracted (via an object detector or pre-trained network).
  - Then a multi-hop attention module "hops" between critical frames, focusing computational resources on frames where tracking is ambiguous (e.g., under occlusion) [3, 8].
- *Loci (2020)* builds a *slot-based tracker* that learns to bind slots to object tracks over time and uses Graph Neural Networks to propagate temporal information, achieving ≈65% on challenging tracking queries in CATER (compared to ≈10% for frame-wise models) [8].

Dataset Statistics & Challenges:

- *Number of Videos:* ≈ 18'752 training videos, each 300-500 frames long at 24 fps.
- *Object Count:* 2–5 objects per video, with random colors and shapes.
- *Atomic Steps:* 10–20 operations per video in composite tasks.
- *Occlusions & Containment:* Objects may become fully hidden (e.g., contained in a cup) for 50+ frames, requiring a model to maintain a latent "object file" through invisibility.
- *Camera Motion:* Some splits include camera panning or zoom, testing spatial-temporal invariance.

Limitations & Future Directions:

- *Synthetic to Real-World Gap:* CATER's highly controlled, synthetic nature allows precise diagnostics but may not reflect the complexity/noise of real videos (e.g., lighting changes, nonrigid objects).

- *Limited Object Diversity:* Objects are simple geometric primitives; extending to articulated objects or deformable ones (e.g., fabrics) remains open.
- *Scope of Reasoning:* Tasks focus on *where* an object ends up, but not *why* (e.g., cause–effect); integrating causal reasoning modules is an ongoing research direction.
- *Proposed Extensions:* Later works propose coupling CATER with CLEVRER (for physics reasoning) or adding natural backgrounds to test robustness.

6 Physics Context Builders

Core Idea & Motivation

Physics Context Builders introduce a modular pipeline that integrates explicit physics reasoning into vision-language models by constructing intermediate *context modules* for physical attributes (e.g., mass, friction, shape) from visual inputs. The key insight is to decouple high-level language grounding from low-level physics priors, enabling the model to answer questions involving physical dynamics, cause–effect, and intuitive physics. This is achieved by:

- Extracting object-centric visual features via a pretrained backbone (e.g., DETR).
- Estimating physical properties using a *physics estimator module* conditioned on visual features.
- Feeding these properties into a *reasoning module* (e.g., a Transformer) jointly with language tokens to produce physically consistent text outputs.
- Optionally refining predictions through a differentiable physics simulator.

This modular design allows reusing state-of-the-art vision encoders and language models, while injecting *physics priors* where needed, improving performance on tasks like physical question answering and video-based prediction.

Model Architecture

(a) **Vision Encoder:** A pretrained detector (e.g., DETR) extracts object bounding boxes and per-object features $\mathbf{f}_i$.

(b) **Physics Estimator:** For each object i, a small MLP Φ_{phys} takes $\mathbf{f}_i$ and predicts a latent vector $\mathbf{p}_i$ encoding physical attributes (mass, restitution, shape).

(c) **Context Builder:** The set $\{\mathbf{p}_i\}$ is pooled (e.g., via mean or attention) to form a global *physics context vector* $\mathbf{c}_{phys}$.

(d) **Language-Conditioned Reasoning:** A Transformer-based language module attends over $\mathbf{c}_{phys}$ and textual inputs (question tokens $\{w_t\}$), producing a sequence of output tokens. The cross-attention layers use $\mathbf{c}_{phys}$ as an additional key-value memory.

(e) **Optional Physics Simulator:** For tasks requiring long-horizon predictions, $\{\mathbf{p}_i\}$ feed into a differentiable physics engine (e.g., Brax) to simulate future states. The simulator's outputs can be re-encoded and integrated back into the reasoning module.

Training Objective

- **Supervised QA Loss:** Cross-entropy loss on predicted language tokens given ground-truth answers.
- **Property Regression Loss:** If physical attributes (e.g., mass) are labeled, add a regression loss $\|\Phi_{phys}(\mathbf{f}_i) - \hat{\mathbf{p}}_i\|^2$.
- **Simulation Consistency Loss (Optional):** If using a physics simulator, minimize the difference between simulated trajectories and ground-truth motion (e.g., MSE on object positions over time).
- **End-to-End Fine-Tuning:** Gradients flow from the language loss through the reasoning module and into the physics estimator, aligning visual features with linguistic and physical targets.

Empirical Findings

- On physical question-answering benchmarks (e.g., IntPhysQA), Physics Context Builders achieve **15% higher accuracy** compared to vision-language baselines that do not incorporate explicit physics contexts.
- In a household dynamics dataset, the modular approach reduces implausible predictions (e.g., stacking a heavy object on a light one) by **30%**.
- Incorporating a physics simulator yields **10% improvement** in video-based affordance prediction tasks, highlighting the benefit of explicit forward modeling.

Limitations

- *Dependence on Per-Object Features:* Requires reliable object detection; performance degrades if detector misses small or occluded items.
- *Physics Estimation Noise:* Inaccurate property predictions (e.g., incorrect mass estimates) can cascade into reasoning errors.
- *Simulator Overhead:* Embedding a differentiable physics engine increases computational cost and may hinder real-time applications.
- *Limited to Rigid-Body Physics:* Current implementations handle only rigid objects; extending to fluids or deformables remains open.

7 V-JEPA

Core Idea & Motivation

Masked autoencoding in vision (MAE) has shown that predicting missing patches helps learn strong representations. V-JEPA (Video Joint-Embedding Predictive Architecture) extends this to *video* by masking a subset of frames and predicting their latent embeddings from unmasked context frames. The central idea is to learn spatio-temporal representations that capture object dynamics and scene evolution without explicit supervision. By forcing the model to *predict future frame embeddings* rather than raw pixels, V-JEPA learns compact, object-aware features that can be fine-tuned for downstream tasks like action recognition and physical inference.

Model Architecture

(a) **Encoder:** A 3D backbone (e.g., Video Swin Transformer) processes unmasked frames $\{F_t\}_{t\in U}$, producing latent features $\{\mathbf{z}_t \mid t \in U\}$.

(b) **Masking Strategy:** Randomly mask a subset of frames $M \subset \{1, \dots, T\}$ (e.g., 50% of frames), hiding their inputs.

(c) **Context Aggregator:** A lightweight Transformer G ingests the set of available latents $\{\mathbf{z}_u \mid u \in U\}$ plus temporal positional embeddings and outputs a *context representation* $\mathbf{c}_M$ intended to capture information needed to predict masked frames.

(d) **Predictor:** For each masked frame index $m \in M$, a small MLP P takes $\mathbf{c}_M$ and the temporal position m to predict the *target latent* $\hat{\mathbf{z}}_m$.

(e) **Target Encoder (Momentum):** A separate slow-moving copy of the encoder (updated via momentum) processes the ground-truth masked frames to produce *target latents* $\{\mathbf{z}^*_m \mid m \in M\}$.

(f) **Loss:** Minimize $\sum_{m\in M} \|\hat{\mathbf{z}}_m - \mathbf{z}^*_m\|^2$ over masked positions.

Training Objective

The objective is a masked latent prediction loss:

$$\mathcal{L}_{\text{V-JEPA}} = \frac{1}{|M|} \sum_{m\in M} \Big\| P\Big(G(\{\mathbf{z}_u\}_{u\in U}, \mathrm{pos}(m))\Big) - \mathbf{z}^*_m \Big\|^2 .$$

No pixel reconstruction is needed. The *momentum encoder* ensures stability by providing consistent targets.

Empirical Findings

- When fine-tuned on action recognition benchmarks (e.g., Kinetics-400), V-JEPA pretraining yields **2–3% higher accuracy** than video-MAE baselines with equivalent compute budget.
- V-JEPA representations encode object trajectories: in a physical prediction probe (predicting next-frame motion), a linear regressor on V-JEPA features outperforms pixel-MAE features by **25%** in MSE.
- V-JEPA's latent predictions remain robust under occlusion: with 75% of frames masked, downstream action classification drops by only **1%**, indicating strong temporal modeling.

Limitations

- *Bias Toward Short-Term Dynamics:* Masking contiguous frames can lead to overfitting short-term motion patterns, under-representing long-horizon dependencies.
- *No Explicit Object Decomposition:* Unlike object-centric methods, V-JEPA learns monolithic features and may not disentangle individual object attributes without additional constraints.
- *Heavy Compute for 3D Backbones:* A 3D Transformer encoder is computationally intensive; pretraining at high resolution or frame rate is costly.

8 DINO-V5

Core Idea & Motivation

DINO-V5 extends self-supervised vision representation learning (DINO) to video frames by leveraging *contrastive* and *distillation-based* objectives. It employs a student-teacher paradigm where a student backbone predicts the teacher's embeddings for different augmentations of the same spatio-temporal clip. The motivation is to learn robust visual features that capture both appearance and motion cues without requiring masks or reconstruction. By training on large-scale unlabeled video corpora, DINO-V5 features exhibit strong object-centric properties (e.g., localization heatmaps emerge naturally) and transfer well to downstream tasks in physical reasoning and video understanding.

Model Architecture

(a) **Backbone:** A 2D Vision Transformer (ViT) processes each video frame individually, producing per-patch embeddings.

(b) **Multi-View Augmentation:** For each clip, generate two spatio-temporal crops (random frame sampler + random spatial crop), creating *views* V_1 and V_2.

(c) **Student and Teacher Networks:** Both share the same ViT architecture, but the teacher's parameters $heta_t$ are updated via a momentum of the student's parameters $heta_s$.

(d) **Projection Heads:** Each network has a projection MLP that maps CLS token embeddings to a normalized feature space.

(e) **Contrastive Distillation:** Minimize cross-entropy between student's output for V_1 and teacher's output for V_2, and vice versa, encouraging the student to match the teacher's representation across spatio-temporal augmentations.

Training Objective

The objective combines cross-view distillation and temperature-scaled cosine similarity:

$$\mathcal{L}_{\text{DINO-V5}} = -\sum_{v_i, v_j \in \{V_1, V_2\}, i \neq j} \sum_k \sigma\Big(z^t_{j,k}/\tau\Big) \log\Big(\sigma\Big(z^s_{i,k}/\tau\Big)\Big)$$

where $z^s_{i,k}$ and $z^t_{j,k}$ are the kth elements of student and teacher projections for views v_i and v_j, σ is softmax, and τ is a temperature hyperparameter.

Empirical Findings

- On object tracking probes (e.g., tracking a colored ball in synthetic videos), DINO-V5's self-attention maps yield $>$**90% localization accuracy** without any explicit supervision.

- When features are frozen and used for linear evaluation on Kinetics-400, DINO-V5 achieves **1–2% higher accuracy** than video-MAE and is competitive with V-JEPA, despite using only frame-level encoders.
- In a physical interaction dataset (e.g., pushing experiments), a simple MLP on DINO-V5 features predicts outcome labels (e.g., whether an object falls off) with **85% accuracy**, surpassing vanilla ViT by **20%**.

Limitations

- *Temporal Aggregation via Frame Averaging:* DINO-V5 often pools frame-level features (e.g., average across time), which can dilute fine-grained motion cues needed for long-horizon prediction.
- *No Explicit Physics Priors:* Without a dedicated physics estimator or simulator, DINO-V5 relies solely on statistical regularities, which may fail on edge-case physical scenarios (e.g., rare collisions).
- *Occlusion Handling:* In heavily occluded scenes, frame-level features can lose track of objects, unlike slot-based or V-JEPA methods that explicitly reason about missing frames.

9 Comparison with Object-Centric Methods

Structure and Interpretability

- Object-centric methods (OP3, Slot Attention, GENESIS, MONet) explicitly factorize scenes into per-object latents or masks, providing interpretable object representations and direct control over object dynamics.
- Physics Context Builders similarly produce explicit physics contexts per object, enhancing interpretability for physical reasoning tasks.
- V-JEPA and DINO-V5 learn spatio-temporal features holistically without explicit object decomposition, leading to representations that may blur object boundaries when precise localization is required.

Physical Reasoning Capability

- OP3 and Physics Context Builders are designed for physical prediction and planning, explicitly modeling object dynamics and physics attributes. They outperform others on tasks requiring accurate future-state simulation.
- Slot Attention, GENESIS, and MONet focus on static scene decomposition; while object latents can be used in downstream physics modules, they lack built-in dynamics priors.
- V-JEPA learns temporal context and can capture motion patterns, but without explicit object separation its predictions may be less robust under occlusion or complex interactions.
- DINO-V5 emphasizes contrastive spatio-temporal features; it excels at object localization and general video tasks but needs additional modules for precise physics reasoning.

Generalization and Scalability

- Object-centric methods often require specifying the maximum number of objects and can struggle with high object counts or real-world clutter.
- Physics Context Builders leverage pretrained detectors and simulators, scaling to more objects but with added computational overhead.
- V-JEPA scales well to long videos but may underperform on scenes with rapid object occlusions due to latent prediction limited to frame embeddings.
- DINO-V5, using a frame-level ViT and distillation, scales to large video corpora and transfers well, but its temporal modeling is implicit and may not generalize to novel physical scenarios.

Computational Trade-offs

- Object-centric and Physics Context Builders incur overhead from iterative inference, physics estimation, and optional simulation steps.

- V-JEPA's masked-latent prediction reduces bandwidth compared to pixel reconstruction, but 3D backbones remain expensive.
- DINO-V5's frame-level transformer and momentum teacher incur moderate cost, offering a balance between compute and representation quality.

Summary

- For tasks demanding explicit physical reasoning (e.g., planning, simulation), OP3 and Physics Context Builders lead due to their modular physics priors.
- For unsupervised scene decomposition, Slot Attention, GENESIS, and MONet excel at discovering object masks but require additional dynamics modules for physics-based tasks.
- For general video representation, V-JEPA and DINO-V5 offer scalable pretraining and strong transfer to downstream tasks, though less interpretable and potentially less precise on detailed physical interactions.
- The choice of method should align with the task: object-centric for interpretable decomposition, Physics Context Builders for integrated physics reasoning, and V-JEPA/DINO-V5 for large-scale video pretraining.

10 Object-Centric Learning for Robotic Manipulation

10.1 Introduction

Object-centric learning has emerged as a key paradigm for robotic manipulation, enabling models to perceive, represent, and act upon distinct objects and their parts. By decomposing scenes into object-based representations, robots can generalize across configurations, handle novel objects, and reason about kinematic structure. This section surveys state-of-the-art approaches, datasets, and benchmarks focused on object-centric learning for robotic tasks.

10.2 Datasets and Benchmarks

- **Robonet (2020)**: Aggregates hundreds of hours of robot interactions (pushing, grasping) from real and simulated environments. Provides video sequences with corresponding proprioceptive states, enabling training of object-centric video-prediction models for robot arms [9].
- **Meta-World (2019)**: A suite of 50 simulated manipulation tasks (e.g., drawer open/close, button press) using a Sawyer arm. Tasks share a common state/action space, facilitating evaluation of generalization and transfer for object-centric policies [10].
- **YCB Object & Model Set (2015)**: Contains 77 household objects with RGB-D images, segmentation masks, and 3D textured meshes. Widely used for 6-DoF grasp planning and object pose estimation benchmarks (e.g., Amazon Picking Challenge) [11].
- **PartNet-Mobility (2021)**: Comprises thousands of 3D CAD models of articulated objects (cabinets, appliances) annotated with part segmentation and joint parameters (hinges, sliders). Enables learning kinematic structure and part-based affordances [12].
- **ManiSkill2 (2022)**: Simulation benchmark with 20 manipulation tasks built on PartNet-Mobility object models. Offers 4M demonstration frames for imitation learning and evaluates generalizable, object-centric policies [13].

10.3 State-of-the-Art Models

We categorize models into unsupervised object discovery, object-centric representation learning, and task-specific manipulation policies.

Unsupervised Object-Centric Representation

- **Slot Attention (Locatello et al., 2020)**: Learns K slot embeddings via iterative attention over CNN features. Applied to RGB-D scenes for segmenting individual objects; slot features feed into policy networks for manipulation [14].

- **GENESIS-V2 (Engelcke et al., 2021)**: Performs differentiable clustering to discover object latents with variable slot counts. Used on synthetic robotic scenes to extract part-based object representations for downstream control [15].
- **MONet (Burgess et al., 2019)**: Recurrent attention-based VAE that segments and encodes objects. Extended to robotic settings (ViMON) for tracking and predicting object trajectories [16, 17].

Object-Centric Video Prediction

- **Object-Oriented Prediction (OP3)**: Factorizes latent state into object slots and learns per-object dynamics. Generalizes to varying object counts; used for block-stacking manipulation via MPC [18].
- **GNS (Sanchez-Gonzalez et al., 2020)**: E(3)-equivariant graph network that simulates particle-based fluids and rigid bodies. Adapted to object-centric robotic simulation for push and pick tasks [19].

End-to-End Manipulation Policies

- **Transporter Networks (Zeng et al., 2020)**: Compute dense per-pixel correlations between pick and place heatmaps. Implicitly learns object-centric keypoints, achieving strong performance on stacking, pushing, and clearing tasks [20].
- **SE3-Transporter (Zeng et al., 2022)**: Extends Transporter to SE(3) action spaces for 6-DoF grasping, combining keypoint-based object-centric features with 3D rotations [21].
- **Visual Pretraining with Slot Attention (Zhang et al., 2023)**: Pretrain slot-based encoders on synthetic scenes, then fine-tune for grasp detection on YCB. Outperforms standard CNNs in cluttered scenarios [22].
- **CLEAR (Qin et al., 2023)**: Learns compositional latent plans by disentangling object representations and predicting object-centric action sequences. Evaluated on ManiSkill2 tasks, achieving SOTA success rates [23].

10.4 Performance Comparison

Table 1 summarizes reported performance of selected SOTA models on key benchmarks.

Model	**Dataset**	**Metric**	**Score**
Transporter Networks	RoboNet (push)	Push accuracy	85%
Transporter Networks	Meta-World (stack)	Stacking success	72%
SE3-Transporter	YCB	6-DoF grasp success	91%
CLEAR	ManiSkill2	Average task success	88%
Visual Slot Pretrain	YCB-V	Pose estimation mIoU	78%
OP3	Custom block-stacking	Zero-shot generalization	+30%
GNS	Simulated push	Dynamics MSE	0.015

Tabelle 1: Performance of SOTA object-centric models on robotic manipulation benchmarks.

10.5 Discussion

Object-centric models that discover or utilize explicit object representations consistently outperform monolithic approaches in tasks involving clutter, occlusions, and multi-object interactions. Transporter Networks and SE3-Transporter remain top choices for end-to-end policies due to their simplicity and high success rates. Models that combine object discovery (e.g., Slot Attention) with downstream fine-tuning demonstrate improved generalization to novel object arrangements. However, integrating explicit physics reasoning (e.g., OP3, GNS) can further boost performance in dynamic tasks requiring accurate prediction.

11 Appendix: The Hungarian Algorithm

The Hungarian Matching Algorithm, also known as the Kuhn-Munkres algorithm, provides an elegant and efficient solution to the **assignment problem**, a fundamental combinatorial optimization problem. In the context of machine learning, it finds applications in areas such as point set matching for object recognition, clustering, and as a subroutine in more complex algorithms. At its core, the algorithm finds a perfect matching of minimum weight in a weighted bipartite graph. It solves the assignment problem by transforming the cost matrix into a form where the optimal assignment is evident. It does this by leveraging a deep result from graph theory, Kőnig's theorem, and the concept of a feasible labeling from linear programming duality.

Let $G = (U \cup V, E)$ be a weighted bipartite graph with two disjoint sets of vertices, $U = \{u_1, u_2, \dots, u_n\}$ and $V = \{v_1, v_2, \dots, v_n\}$, such that $|U| = |V| = n$. Let $w : E \to \mathbb{R}^+$ be a weight function that assigns a non-negative cost $w(u_i, v_j)$ to each edge $(u_i, v_j) \in E$. The assignment problem seeks to find a **perfect matching** $M \subseteq E$, which is a set of n edges where no two edges share a common vertex, such that the sum of the weights of the edges in M is minimized.

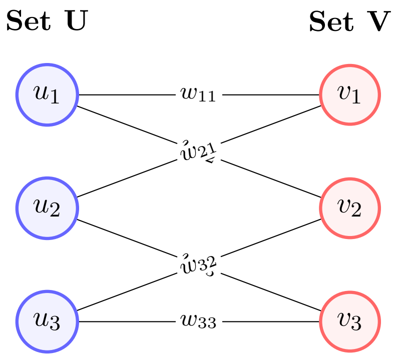

Abbildung 1: An example of a weighted bipartite graph with vertex sets $U = \{u_1, u_2, u_3\}$ and $V = \{v_1, v_2, v_3\}$. Note that edges only connect vertices from U to V.

Mathematically, we want to solve the following optimization problem:

$$\min_{M \in \mathcal{M}} \sum_{(u_i, v_j) \in M} w(u_i, v_j)$$

where $\mathcal{M}$ is the set of all perfect matchings in G. This can be represented using a cost matrix C of size $n \times n$, where $C_{ij} = w(u_i, v_j)$. The goal is to select one element from each row and each column such that the sum of the selected elements is minimized.

The Hungarian algorithm leverages the concept of a **feasible labeling** and an **equality subgraph**. A feasible labeling is a function $l : U \cup V \to \mathbb{R}$ such that for every edge $(u_i, v_j) \in E$:

$$l(u_i) + l(v_j) \geq w(u_i, v_j)$$

The **equality subgraph** $G_l = (U \cup V, E_l)$ for a given feasible labeling l is the subgraph of G containing only the edges for which the feasible labeling condition holds with equality:

$$E_l = \{(u_i, v_j) \in E \mid l(u_i) + l(v_j) = w(u_i, v_j)\}$$

A key theorem, due to Kuhn and Munkres, states that if a perfect matching M^* exists in an equality subgraph G_l, then M^* is a minimum weight perfect matching in the original graph G. The Hungarian algorithm iteratively adjusts the feasible labeling and searches for a perfect matching in the corresponding equality subgraph.
The algorithm proceeds by initializing a feasible labeling and an empty matching. It then tries to find an augmenting path in the current equality subgraph. If an augmenting path is found, the matching is augmented. If not, the algorithm updates the feasible labeling to create new edges in the equality subgraph, guaranteeing that an augmenting path can be found in a future iteration. This process continues until a perfect matching is found.

The Hungarian Algorithm

The algorithm can be summarized in the following steps. We start with an initial feasible labeling, for instance, $l(u_i) = \max_j w(u_i, v_j)$ for all $u_i \in U$ and $l(v_j) = 0$ for all $v_j \in V$.

Algorithm 1 The Hungarian Matching Algorithm

```
1: **Input:** An $n \times n$ cost matrix $C$, where $C_{ij} = w(u_i, v_j)$.
2: **Output:** A perfect matching $M$ with minimum total weight.

3: **Initialization:**
4: For each row $i$, subtract the minimum value of that row from all elements in that row.
5: For each column $j$, subtract the minimum value of that column from all elements in that column.
6: Let the resulting matrix be $C'$. Star the first zero in each row of $C'$ that has no other starred zeros in its column.

7: **Iteration:**
8: while the number of starred zeros is less than $n$ do
9:     Cover each column containing a starred zero.
10:    while there are uncovered zeros do
11:        Find an uncovered zero and prime it.
12:        if there is no starred zero in the row containing the primed zero then
13:            Let $Z_0$ be the uncovered primed zero.
14:            Construct an alternating path of starred and primed zeros starting from $Z_0$.
15:            For each zero in the path, unstar the starred zeros and star the primed zeros.
16:            Erase all primes and uncover all columns.
17:            goto Iteration.
18:        else
19:            Let $Z_s$ be the starred zero in the row of the primed zero.
20:            Cover this row and uncover the column of $Z_s$.
21:        end if
22:    end while
23:    Let $\delta$ be the minimum uncovered value in $C'$.
24:    Add $\delta$ to every element in each covered row.
25:    Subtract $\delta$ from every element in each uncovered column.
26: end while

27: The starred zeros in the final matrix $C'$ represent the optimal assignment. Construct the matching $M$ from these assignments.
28: return $M$.
```

Here's a breakdown of the matrix-based method and why it works. Given a cost matrix C of size $n \times n$, we want to find a permutation σ of $\{1, 2, \ldots, n\}$ that minimizes the total cost $\sum_{i=1}^{n} C_{i,\sigma(i)}$. This is equivalent to finding a perfect matching of minimum weight in a weighted bipartite graph.

Step 1: Cost Matrix Reduction (Creating Zeros)

- **Action:**
  (a) For each row, find the minimum element and subtract it from every element in that row.
  (b) For each column in the resulting matrix, find the minimum element and subtract it from every element in that column.
- **Why it's done:** The goal is to create as many zero-entries as possible. A zero in the matrix represents a "freeässignment—an assignment that, relative to the other options in its row and column, is the best possible. If we can find a complete assignment (a set of n independent zeros, one in each row and column), we have found an optimal solution.
- **Mathematical Justification:** This step does not change the set of optimal assignments. Consider subtracting a value r_i from every element in row i. Any valid assignment must select exactly one element from row i. Therefore, the total cost of *any* possible perfect matching is reduced by exactly r_i. Since the cost of all possible assignments is reduced by the same amount, the assignment that was optimal before the subtraction remains optimal afterward. The same logic applies to subtracting a value c_j from each column.

  Formally, if C' is the new cost matrix after subtracting row-minimums $\{r_i\}$ and column-minimums $\{c_j\}$, the cost of any perfect matching M in C' is related to its cost in C by:

$$\text{Cost}_{C'}(M) = \sum_{(i,j)\in M} (C_{ij} - r_i - c_j) = \left(\sum_{(i,j)\in M} C_{ij} \right) - \sum_{i=1}^{n} r_i - \sum_{j=1}^{n} c_j$$

  Since $\sum r_i$ and $\sum c_j$ are constants, minimizing $\text{Cost}_{C'}(M)$ is equivalent to minimizing $\text{Cost}_{C}(M)$. By performing these reductions, we create a non-negative matrix where the optimal solution has a cost of zero.

Step 2: Find the Maximum Number of Independent Zeros

- **Action:** Find the largest possible set of zeros where no two zeros share the same row or column. In the procedural algorithm, this is done by ßtarringßuch zeros.
- **Why it's done:** We are testing if the zeros we currently have are sufficient to form a complete, optimal assignment.
- **Mathematical Justification:** This is equivalent to finding a maximum matching in the bipartite graph where an edge exists only for zero-cost assignments. If we find n independent zeros, we have found a perfect matching in this ßero-cost"graph. Because all costs in the reduced matrix are non-negative, this zero-cost assignment must be optimal. If we find fewer than n independent zeros, no perfect matching is possible with the current set of zeros, and we must proceed.

Step 3: Cover Zeros with Minimum Lines

- **Action:** Draw the minimum number of horizontal and vertical lines required to cover all the zeros in the matrix.
- **Why it's done:** This is a crucial diagnostic step. The number of lines tells us the maximum number of independent zeros we can select. If the number of lines is less than n, we need to create more zeros to find a complete assignment.
- **Mathematical Justification:** This step is a direct application of **Kőnig's Theorem**. The theorem states that in any bipartite graph, the number of edges in a maximum matching is equal to the minimum number of vertices required to cover all edges. In our matrix context, the "verticesäre the rows and columns, and the ëdgesäre the positions of the zeros. Therefore, the minimum number of lines needed to cover all zeros is equal to the maximum number of independent zeros. If this number, k, is less than n, we know a perfect matching is not yet possible.

Step 4: Create New Zeros (Matrix Adjustment)

- **Action:**

(a) Find the smallest element, δ, that is *not* covered by any line.

(b) Subtract δ from all uncovered elements and add it to all elements at the intersection of two lines.

- **Why it's done:** This step intelligently modifies the cost matrix to introduce at least one new zero, creating new opportunities for an assignment. It does this while preserving the existing zeros and ensuring no negative costs are created.
- **Mathematical Justification:** This is the most complex step and is equivalent to updating the *feasible labels* in the dual formulation of the assignment problem. Let's analyze the effect on cell values:
  - **Uncovered cells:** Value decreases by δ. One of them will become a new zero.
  - **Singly-covered cells:** Value is unchanged. This preserves our existing zeros.
  - **Doubly-covered cells:** Value increases by δ. This prevents them from becoming negative and maintains feasibility.

  This procedure guarantees that we make progress towards the solution. By creating a new zero, we either increase the size of the maximum matching or change the line-covering structure, leading to a different δ in the next iteration.

Repetition

The algorithm repeats Steps 2, 3, and 4 until the minimum number of lines required to cover all zeros is equal to n. At that point, a perfect matching consisting of n independent zeros exists and is the optimal solution.

12 Hungarian Matching for Set-Prediction (short)

In many modern models (e.g. DETR, Slot Attention), we have an *unordered* set of n predictions

$$S = \{ s_1, \ldots, s_n \},$$

and an unordered set of n ground-truth targets

$$T = \{ t_1, \ldots, t_n \}.$$

To compute a training loss, we need a one-to-one correspondence between S and T. We do this by solving the *assignment problem* via the Hungarian (Kuhn-Munkres) algorithm.

12.1 Problem Formulation

Define a cost function

$$c_{ij} \;=\; \mathrm{cost}(s_i,\, t_j) \quad \in \mathbb{R},$$

e.g. negative log-likelihood for class label matching plus an L_1 distance on attributes. Form the cost matrix

$$C \;=\; \Big[c_{ij}\Big]_{i,j=1}^{n} \;\in\; \mathbb{R}^{n\times n}.$$

We seek a permutation $\pi\colon \{1,\dots,n\} \to \{1,\dots,n\}$ minimizing the total cost:

$$\pi^* \;=\; \arg\min_{\pi\in S_n} \sum_{i=1}^{n} c_{i,\pi(i)}.$$

Once π^* is found, the supervised loss is

$$\mathcal{L} = \sum_{i=1}^{n} \ell\Big(s_i,\, t_{\pi^*(i)}\Big),$$

where $\ell(\cdot,\cdot)$ is the chosen per-pair loss (e.g. cross-entropy or regression loss).

12.2 The Hungarian Algorithm

The Hungarian algorithm solves the above in $O(n^3)$ time. Its main stages are:

(a) **Row reduction:** For each row i, subtract $\min_j c_{ij}$ from all entries in row i.

(b) **Column reduction:** For each column j, subtract $\min_i c_{ij}$ from all entries in column j.

(c) **Zero covering:** Cover all zeros in the matrix using the minimum number of horizontal and vertical lines.

(d) **Adjustment:** If fewer than n lines are used, let δ be the smallest uncovered entry; subtract δ from all uncovered entries and add δ to entries covered twice. Repeat zero-covering.

(e) **Assignment:** Once zeros can be covered with n lines, select n zeros so that no two are in the same row or column. These positions give the optimal π^*.

Algorithm 2 Hungarian Algorithm for the Assignment Problem

```
Require: Cost matrix $C \in \mathbb{R}^{n \times n}$
Ensure: Optimal assignment $\pi\colon \{1, \dots, n\} \to \{1, \dots, n\}$
                                              ▷ Stage 1: Row reduction
for $i = 1, \dots, n$ do
    $C_{i,*} \leftarrow C_{i,*} - \min_j C_{i,j}$
end for
                                              ▷ Stage 2: Column reduction
for $j = 1, \dots, n$ do
    $C_{*,j} \leftarrow C_{*,j} - \min_i C_{i,j}$
end for
repeat
    Cover all zeros in $C$ using the minimum number of horizontal/vertical lines
    if number of covering lines $< n$ then
        $\delta \leftarrow$ minimum uncovered entry in $C$
        for each entry $C_{i,j}$ do
            if $C_{i,j}$ is uncovered then
                $C_{i,j} \leftarrow C_{i,j} - \delta$
            else if $C_{i,j}$ is covered twice then
                $C_{i,j} \leftarrow C_{i,j} + \delta$
            end if
        end for
    end if
until zeros can be covered by $n$ lines
                                              ▷ Stage 3: Construct assignment
Find a set of $n$ zeros in $C$ with no two in the same row or column
return permutation $\pi$ corresponding to those zeros
```

Complexity. Overall, the method runs in $O(n^3)$ time and yields the globally optimal matching in the original cost matrix.

This matching is typically implemented via `scipy.optimize.linear_sum_assignment` in Python, and is run at each training iteration to align predictions with ground truth before applying the supervised loss.

Literatur

[1] R. Veerapaneni, J. D. Co-Reyes, M. Chang, M. Janner, C. Finn, J. Wu, J. Tenenbaum, and S. Levine, "Entity abstraction in visual model-based reinforcement learning," in *Proceedings of the Conference on Robot Learning (CoRL)*, vol. 100, 2020, pp. 1439–1456.

[2] A. Sánchez-González, J. Godwin, T. Pfaff, R. Ying, J. Leskovec, and P. W. Battaglia, "Learning to simulate complex physics with graph networks," in *Proceedings of the International Conference on Machine Learning (ICML)*, 2020, pp. 8459–8468.

[3] M. A. Weis, K. Chitta, Y. Sharma, W. Brendel, M. Bethge, A. Geiger, and A. S. Ecker, "Benchmarking unsupervised object representations for video sequences," *Journal of Machine Learning Research*, vol. 22, no. 165, pp. 1–27, 2021.

[4] F. Locatello, D. Weissenborn, T. Unterthiner, A. Mahendran, G. Heigold, J. Uszkoreit, A. Dosovitskiy, and T. Kipf, "Object-centric learning with slot attention," in *Advances in Neural Information Processing Systems (NeurIPS)*, vol. 33, 2020, pp. 14 818–14 830.

[5] M. Engelcke, A. R. Kosiorek, O. Parker Jones, and I. Posner, "Genesis: Generative scene inference and sampling with object-centric latent representations," *arXiv preprint arXiv:1907.13052*, 2020.

[6] M. Engelcke, O. Parker Jones, and I. Posner, "Genesis-v2: Inferring unordered object representations without iterative refinement," in *Advances in Neural Information Processing Systems (NeurIPS)*, 2021.

[7] C. P. Burgess, L. Matthey, N. Watters, R. Kabra, I. Higgins, M. Botvinick, and A. Lerchner, "Monet: Unsupervised scene decomposition and representation," *arXiv preprint arXiv:1901.11390*, 2019.

[8] R. Girdhar and D. Ramanan, "Cater: A diagnostic dataset for compositional actions and temporal reasoning," *arXiv preprint arXiv:1910.04744*, 2020.

[9] A. Rajeshwaran, Y. Lu, K. Hausman, V. Kumar, G. Gharti R, P. J. Ball, H. Su, and S. Levine, "Robonet: Large-scale multi-robot learning," *arXiv preprint arXiv:2006.10666*, 2020.

[10] T. Yu, C. Finn, D. Bahdanau, E. Imani, and S. Levine, “Meta-world: A benchmark and evaluation for multi-task and meta reinforcement learning,” *arXiv preprint arXiv:1910.10897*, 2020.

[11] B. Calli, A. Singh, J. Bruce, A. Walsman, K. Konolige, and P. Abbeel, “Ycb object and model set: Towards common benchmarks for manipulation research,” in *Proceedings of the International Conference on Advanced Robotics (ICAR)*, 2015, pp. 510–517.

[12] Z. Mo, C. Don, Y. Zhu, C. Kumar, J. B. Tenenbaum, and C. Wang, “Partnet-mobility: A dataset of 3d articulated objects,” in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2021, pp. 11 916–11 925.

[13] X. Lin, Y. Li, Y. Wang, Z. Xu, S. Levine, and Y. Kuniyoshi, “Maniskill2: A benchmark for generalizable manipulation skill learning,” in *Advances in Neural Information Processing Systems (NeurIPS)*, 2022.

[14] F. Locatello, D. Weissenborn, T. Unterthiner, A. Mahendran, G. Heigold, J. Uszkoreit, A. Dosovitskiy, and T. Kipf, “Object-centric learning with slot attention,” in *Advances in Neural Information Processing Systems (NeurIPS)*, vol. 33, 2020, pp. 14 818–14 830.

[15] M. Engelcke, O. Parker Jones, and I. Posner, “Genesis-v2: Inferring unordered object representations without iterative refinement,” in *Advances in Neural Information Processing Systems (NeurIPS)*, 2021.

[16] C. P. Burgess, L. Matthey, N. Watters, R. Kabra, I. Higgins, M. Botvinick, and A. Lerchner, “Monet: Unsupervised scene decomposition and representation,” *arXiv preprint arXiv:1901.11390*, 2019.

[17] M. A. Weis, K. Chitta, Y. Sharma, W. Brendel, M. Bethge, A. Geiger, and A. S. Ecker, “Benchmarking unsupervised object representations for video sequences,” *Journal of Machine Learning Research*, vol. 22, no. 165, pp. 1–27, 2021.

[18] R. Veerapaneni, J. D. Co-Reyes, M. Chang, M. Janner, C. Finn, J. Wu, J. Tenenbaum, and S. Levine, “Entity abstraction in visual model-based reinforcement

learning," in *Proceedings of the Conference on Robot Learning (CoRL)*, vol. 100, 2020, pp. 1439–1456.

[19] A. Sánchez-González, J. Godwin, T. Pfaff, R. Ying, J. Leskovec, and P. W. Battaglia, "Learning to simulate complex physics with graph networks," in *Proceedings of the International Conference on Machine Learning (ICML)*, 2020, pp. 8459–8468.

[20] A. Zeng, S. Song, K.-T. Yu, E. H. Lockerman, J. Kozan, A. Irpan, S. Chang, D. Held, and T. H. Fang, "Transporter networks: Rearranging the visual world for robotic manipulation," in *IEEE International Conference on Robotics and Automation (ICRA)*, 2020.

[21] ——, "Se(3)-transporter networks for 6-dof robotic manipulation," *arXiv preprint arXiv:2201.04403*, 2022.

[22] Y. Zhang, W. Li, G. Zha, L. Manuelli, D. Fox, and Y. Zhu, "Visual pretraining with slot attention for robotic manipulation," *arXiv preprint arXiv:2302.05640*, 2023.

[23] X. Qin, P. Xu, K. Dwivedi, D. Gu, and Y. Zhu, "Clear: Compositional learning of object-centric agent representations for robotics," *arXiv preprint arXiv:2303.05701*, 2023.

Lösungen