Appendix A. Trackastra Backbone

We summarize key components of the *Trackastra* [5] model below. *Trackastra* is a transformer-based architecture trained in a supervised manner to perform cell tracking across time-lapse microscopy data. During training, it assumes all cell detections are correct and focuses exclusively on learning associations between them.

The model processes overlapping spatio-temporal windows of size T frames$\times H \times W$ pixels. Each detection within this window is tokenized based on low-level morphological features—specifically, mean intensity, object area, and inertia tensor. To construct input tokens, these features are concatenated with learned Fourier spatial positional encodings, which capture both spatial and temporal context. The resulting vector is then projected onto a fixed token dimension, embedding each detection with information about its appearance, position, and time.

The model outputs a square association matrix of shape $N \times N$, where N is the total number of detections in the window. A *quiet parental-softmax* is applied to the matrix to enforce that each detection may be assigned to at most one parent in the previous frame—or none, thereby naturally accommodating appearing and disappearing objects.

Supervision is provided in the form of a binary ground-truth association matrix, where the entry is 1 if two detections belong to the same cell lineage (*i.e.*, one is a direct ancestor or descendant of the other), and 0 otherwise.

The supervised loss used by *Trackastra* is a weighted combination of two binary cross-entropy losses:

$$\mathcal{L}_{\text{sup.}}(A, \hat{A}, W) = \mathcal{L}_{\text{BCE}}\left(A, \Phi(\hat{A}), W\right) + 0.01\, \mathcal{L}_{\text{BCE}}\left(A, \sigma(\hat{A}), W\right), \tag{8}$$

where $\Phi(\hat{A})$ is the quiet-softmax output, $\sigma(\hat{A})$ is the element-wise sigmoid applied to the predicted association matrix $\hat{A}$, and W is a weighting factor for each element.

In this paper, we use $\mathcal{A}$ instead of $\Phi(\hat{A})$ to denote the quiet-softmax output.

For full implementation details, we refer the reader to *Trackastra* [5] .

Appendix B. Evaluation Metrics

We evaluate the performance of our proposed method and baseline approaches using several standard tracking metrics, as reported in Table 1, Table 2, Figure 2 and Figure 3. All metrics are computed using the *Traccuracy* library available at https://github.com/live-image-tracking-tools/traccuracy.

Acyclic Oriented Graph Matching (AOGM [13]) quantifies the effort required to transform a predicted tracking graph into the ground truth graph. An AOGM of zero indicates a perfect match. In the context of linking ground truth detections, it is computed as a weighted sum of node and edge errors:

$$\text{AOGM} = 10 \cdot |\text{FN}_\text{n}| + |\text{FP}_\text{e}| + 1.5 \cdot |\text{FN}_\text{e}| + |\text{WS}_\text{e}|,$$

where $|\text{FN}_\text{n}|$ denotes the number of false negative nodes, $|\text{FP}_\text{e}|$ the number of false positive edges, $|\text{FN}_\text{e}|$ the number of false negative edges, and $|\text{WS}_\text{e}|$ the number of edges with wrong semantics (e.g., division vs. linking).

Tracking Accuracy (TRA) [13] normalizes AOGM to the $[0, 1]$ range, enabling consistent comparisons across datasets:

$$\text{TRA} = 1 - \frac{\min\left(\text{AOGM}, \text{AOGM}_0\right)}{\text{AOGM}_0},$$

where AOGM_0 is the cost of transforming an empty graph (no nodes or edges) into the ground truth.

Since $\text{AOGM} \downarrow$ and $\text{TRA} \uparrow$ only implicitly account for cell divisions, we also report dedicated cell division metrics, some of which are described below..

Mitotic Branching Correctness (MBC) [2] measures the fraction of correctly detected mitotic events out of all mitotic events in the ground truth. A mitosis is considered correct if the predicted and true division times of occurrence are within a specified tolerance, and the daughter assignments match.

Division F_1 Score (Div. F_1) evaluates division detection using the counts of true positive TP_d, false positive FP_d, and false negative FN_d division events:

$$\text{Div. F}_1 = \frac{2 \cdot \text{TP}_\text{d}}{2 \cdot \text{TP}_\text{d} + \text{FP}_\text{d} + \text{FN}_\text{d}}.$$

Appendix C. Experimental Setup

Data Handling During unsupervised pre-training, we use the entire dataset—without a train/val/test split—to maximize exposure to unlabeled data. For the *Bacteria* [5, 19] dataset, this includes all 30 training, 2 validation, and 6 test sequences. Evaluation is still performed only on the 6 test sequences. For the *HeLa* [12] dataset, both sequences (01 and 02) are used during pre-training, but we evaluate exclusively on sequence 02.

During fine-tuning:

- In the *extrapolation* setup, we fine-tune using annotations derived from both train and validation sequences.
- In the *interpolation* setup, we fine-tune directly on the test sequences.

Evaluation is always conducted on the test sequences in both setups.

Evaluation Protocol For the *Bacteria* [5, 19] dataset, which contains 6 test video sequences, evaluation metrics are computed independently for each sequence and then averaged, following the same protocol as *Trackastra* [5] .

Globally-Constrained Optimization In all experiments, we use the following cost configuration: $w_1{=}w_2{=}1$, $b{=}0$, and fixed costs $C_{\mathrm{a}}{=}C_{\mathrm{d}}{=}1$ for appearance and disappearance events. All distance measures d_1 and d_2 are normalized to have zero mean and unit variance, separately for standard edges and hyper edges.

Supervised Baseline For the supervised baseline (see *Position and Supervised Associations* in Table 1), we use the first 78 frames of sequence 01 in the *HeLa* [12] dataset for training, and the last 14 frames are held out for validation. Similar to above, evaluation is performed on the sequence 02 of the *HeLa* dataset.

In this baseline, we train a *Trackastra* [5] model in a supervised manner, using the provided ground truth association matrices and the supervised loss function (see Equation 8). Next, in Equation 5, we set $d_2 = -\mathcal{A}_{i,j}^{t-1,t}$ where $\mathcal{A}^{t-1,t}$ is the predicted association matrix output, by the model between cell detections corresponding to frames $t{-}1$ and t while the subscript (i,j) denotes the score for the edge between cell detections i and j at frames $t{-}1$ and t, respectively.

Re-ID and Autoencoder Baselines For the Re-ID and autoencoder-based baselines, we use patches of size $64{\times}64$ pixels. The autoencoder produces a 64-dimensional embedding for each patch, which is then used as input for the tracker trained with unsupervised loss (see Equation 3).

***Attrackt* Pre-Training** We set the weight of the transitive loss in Equation 3 to $\lambda = 0.1$. Our tracking backbone is based on *Trackastra* [5] using 6 encoder and 6 decoder layers. We train all models for 600 epochs with a batch size of 8, a dropout rate of 0.1, and a spatio-temporal window of size 6 frames$\times 256 \times 256$ pixels.

We use lightweight MLPs, denoted m_{E} and m_{D} (see Figure 1 and Section 3.2), to process autoencoder embeddings in a parallel branch to the *Trackastra* [5] network. These MLPs follow the structure below:

```
class FeedForward(torch.nn.Module):
    def __init__(self, d_model, expand: float = 2, bias: bool = True):
        super().__init__()
        self.fc1 = torch.nn.Linear(d_model, int(d_model * expand))
        self.fc2 = torch.nn.Linear(int(d_model * expand), d_model, bias=bias)
        self.act = torch.nn.GELU()
        self.norm = torch.nn.LayerNorm(d_model)

    def forward(self, x):
        return x + self.fc2(self.act(self.fc1(self.norm(x))))
```

***Attrackt* Fine-Tuning** During LoRA-based fine-tuning, we set the rank $r = 32$ and the scaling factor $\alpha = 32$.

To prevent overfitting during fine-tuning, we additionally apply an early stopping criterion: training is halted after $100 \cdot k$ iterations, and the model checkpoint from the final iteration is used for inference, where k is the number of node annotations.

Residual Error in Interpolation In principle, since we incorporate *pinning* (i.e., enforcing known annotations while solving the globally-constrained optimization) in the *interpolation* setting, the $\mathrm{AOGM} \downarrow$ should drop to zero when all annotations are used. However, we observe a small residual error (see Figures 2 and 3). This occurs because the candidate graph connects each detection to its 10 nearest neighbors in the previous frame. Occasionally, the true predecessor lies outside this neighborhood.

Appendix D. Comparison of pre-training followed by fine-tuning vs. Training from scratch

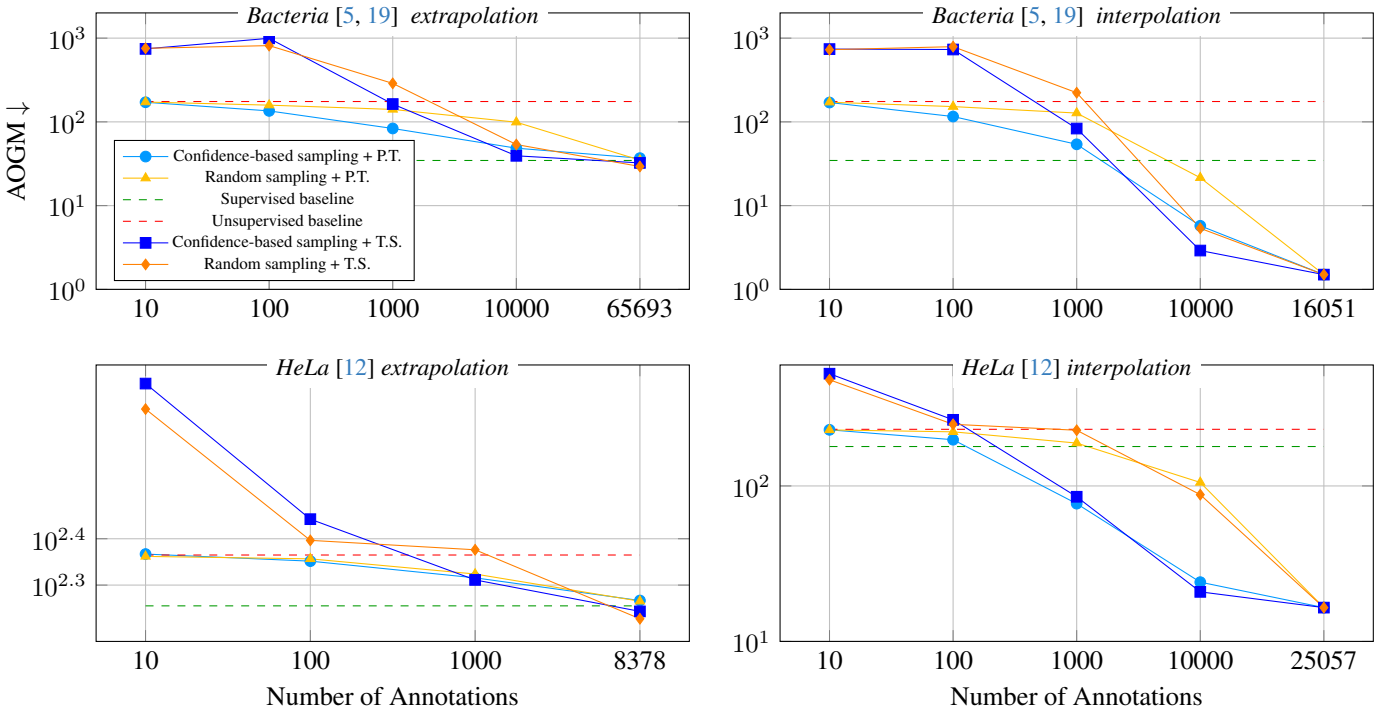

Figure 3. The figure shows a quantitative comparison of unsupervised pre-training (P.T.) (using *Attrackt* and transitive loss, see Equation 3) followed by fine-tuning, versus supervised training from scratch (T.S.), with a limited number of annotations on two microscopy datasets: **Bacteria** and **HeLa**. Each row corresponds to a dataset, and plots the AOGM ↓ metric after providing k ground truth (G.T.) annotations. Each column corresponds to an experimental setup: *Extrapolation* refers to training with G.T. annotations from the train+val split and evaluating on the test split, while *Interpolation* refers to providing G.T. annotations directly on the test split, which is also used for evaluation. Results using two sampling strategies (confidence-based and random) are shown for both P.T. and T.S. *Supervised* and *Unsupervised* baselines correspond to results with *Pos. & Sup. Associations* and *Pos. & Unsup. Associations* in Table 1.