# Pose Tracking *Eisenia fetida* with Deep Learning Worm Trackers

## Pretrained Models vs. Large Worm Morphology

Most deep-learning worm trackers were developed and **pretrained on *C. elegans***, a much smaller and thinner nematode. This means their models learned worm features (shape, size, movement) specific to *C. elegans*. For a larger, thicker worm like *Eisenia fetida*, the core algorithms still apply, but some adjustments may be needed:

* **Deep-Worm-Tracker (YOLO + SORT):** This tracker’s detection network was trained on ~3000 images of *C. elegans* worms[[1]](https://github.com/knaticat/Deep-Worm-Tracker#:~:text=,the%20YOLO%20object%20detection%20model). It should recognize an elongated worm shape even if the worm is larger – YOLO is somewhat scale-invariant – but the **appearance differences** (thicker body, possibly different texture or contrast) could affect confidence. The original authors noted that their model struggled with out-of-distribution worm sizes (it had **limitations detecting very small worms**, for instance)[[2]](https://www.nature.com/articles/s41598-025-93533-0?error=cookies_not_supported&code=45077ede-dc61-41c4-9bd3-b1a52daebcaf#:~:text=tracking%20methods%20has%20also%20emerged,training%20times%2C%20and%20slow%20inference). By extension, a much *larger* worm might also be outside the model’s comfort zone. In practice, the pretrained YOLO might still detect an *E. fetida* as a “worm” object on a plain background, but **accuracy may not be optimal** without fine-tuning. Fine-tuning the detector on a handful of *E. fetida* frames (annotating worm bounding boxes) would adapt the model to the worm’s thicker profile. The SORT tracking part (maintaining worm identity) is trivial here since you have only one worm. In summary, the Deep-Worm-Tracker **pretrained weights can be used**, but expect to possibly recalibrate the detection threshold or retrain on a small sample for best results. The good news is that training a YOLO model on a new dataset is fast (on the order of minutes)[[3]](https://github.com/knaticat/Deep-Worm-Tracker#:~:text=change%20in%20magnification%2C%20dust%20particles,worm%20trajectories%20are%20also%20highlighted), so adapting it to your worms is feasible if needed.
* **WormPose:** WormPose is an open-source pose estimator originally developed for *C. elegans* that uses a convolutional neural network (CNN) to predict the worm’s centerline **even in coiled or self-occluded poses**[[4]](https://journals.plos.org/ploscompbiol/article?id=10.1371/journal.pcbi.1008914#:~:text=animals%20such%20as%20the%20nematode,food). It was designed for single-worm scenarios and actually requires a preliminary tracking step (e.g. using Tierpsy) to supply initial centerlines for non-coiled frames[[2]](https://www.nature.com/articles/s41598-025-93533-0?error=cookies_not_supported&code=45077ede-dc61-41c4-9bd3-b1a52daebcaf#:~:text=tracking%20methods%20has%20also%20emerged,training%20times%2C%20and%20slow%20inference). Importantly, WormPose was built to be **adaptable across imaging conditions**[[4]](https://journals.plos.org/ploscompbiol/article?id=10.1371/journal.pcbi.1008914#:~:text=animals%20such%20as%20the%20nematode,food). The authors explicitly state that while it’s tuned on *C. elegans*, the method (synthetic training of a worm-shape model) should generalize to *“other slender-bodied and deformable organisms.”*[[5]](https://journals.plos.org/ploscompbiol/article?id=10.1371/journal.pcbi.1008914#:~:text=misidentified%20as%20an%20increase%20in,bodied%20and%20deformable%20organisms) In other words, a larger worm like *E. fetida* can be handled, *provided you train the model on data reflective of that worm*. There isn’t a ready-made pretrained weight for *Eisenia* worms; you would generate a training set (using your videos) and train a WormPose model. This involves using your video frames to create synthetic worm images matching *E. fetida*’s size/shape. You can specify the worm’s approximate width in the WormPose configuration so that the synthetic data matches the thick radius of your worms (the pipeline draws a worm outline of a given width along the centerline)[[6]](https://journals.plos.org/ploscompbiol/article?id=10.1371/journal.pcbi.1008914#:~:text=protrusions%20are%20still%20visible%2C%20especially,the%20desired%20worm%20width%2C%20complete). In summary, **WormPose can support larger worms**, but you will need to go through its training pipeline with your data. The approach is quite general once trained: WormPose has been shown to work across different substrates and imaging setups for nematodes[[7]](https://journals.plos.org/ploscompbiol/article?id=10.1371/journal.pcbi.1008914#:~:text=An%20important%20model%20system%20for,synthetic%20data%20as%20well%20as), so differences in worm size and thickness can be learned by the model as long as you provide representative training examples.
* **Other Pose Estimation Models:** Aside from those two, general animal pose estimators like **DeepLabCut or SLEAP** could be applied to worms. They don’t have worm-specific pretraining, so you would need to label keypoints on your worms and train them. The benefit is that these frameworks are species-agnostic – a few dozen labeled frames of *E. fetida* with points along the body could yield a model that predicts worm poses. However, this is a manual effort. There are also newer worm-specific models in research (e.g. **DeepTangleCrawl (DTC)** in 2025) that explicitly target complex worm poses, but these aren’t off-the-shelf solutions. DTC, for example, was trained on *C. elegans* data that included many overlaps and coils, and it outperforms earlier trackers in those scenarios[[8]](https://journals.plos.org/ploscompbiol/article?id=10.1371/journal.pcbi.1013345#:~:text=have%20addressed%20these%20issues%20to,screens%2C%20even%20for%20data%20that). Its code is available, but it might require integration with the Tierpsy framework. If you’re looking for a pretrained **turnkey** solution, Deep-Worm-Tracker and WormPose are the primary candidates, with the understanding that WormPose will need training on your worm’s data and Deep-Worm-Tracker may need minor retraining or calibration.

## Handling Self-Overlap and Coiling Behaviors

Challenging poses like self-overlaps, partial/full coils, and omega bends are exactly where many trackers struggle. Classic image-processing trackers (e.g. Tierpsy or OpenCV-based skeletonizers) often **fail when a worm overlaps itself**, because the worm’s outline no longer looks like a simple open curve – the body touches or crosses over, confounding standard skeleton algorithms. In fact, the Tierpsy tracker tends to drop the skeleton when a worm self-intersects, creating data gaps during coiled postures[[9]](https://journals.plos.org/ploscompbiol/article?id=10.1371/journal.pcbi.1013345#:~:text=When%20worms%20perform%20sharp%20turns%2C,most%20characteristic%20so%20missing%20them). Your question specifically targets whether Deep-Worm-Tracker or WormPose can handle these events (and if fine-tuning helps), so let’s consider each:

* **Deep-Worm-Tracker:** This system uses object detection (YOLO) to localize worms and a tracking algorithm (StrongSORT) to maintain identities. On a single worm, identity is a given, so the crucial part is detection and what happens after detection. Notably, Deep-Worm-Tracker does **not** explicitly model the worm’s shape or pose with a deep network – it can output a segmentation and skeleton, but those are obtained via traditional methods (thresholding the image and then skeletonizing)[[3]](https://github.com/knaticat/Deep-Worm-Tracker#:~:text=change%20in%20magnification%2C%20dust%20particles,worm%20trajectories%20are%20also%20highlighted). During a self-overlap or omega bend, YOLO will still correctly detect the worm as one object (since the worm is contiguous in the image). So you will **not lose track of the worm** – the bounding box will still cover the worm’s whole body even if folded or coiled. However, the **accuracy of the skeleton it produces in that frame is not guaranteed**. Standard skeletonization might, for example, generate a **broken or doubled skeleton** when the worm is looped on itself (imagine a pretzel shape – a simple thinning algorithm can’t discern the correct one-pixel-wide centerline easily). The Deep-Worm-Tracker paper focused on multi-worm tracking and reported high detection/tracking precision on *C. elegans*, but it did not specifically demonstrate success on self-occluded postures like omega turns. In fact, newer research that compared methods found that Deep-Worm-Tracker performed significantly worse on **strongly overlapping worm images** than approaches explicitly designed for overlaps[[10]](https://www.nature.com/articles/s41598-025-93533-0?error=cookies_not_supported&code=45077ede-dc61-41c4-9bd3-b1a52daebcaf#:~:text=On%20three%20datasets%2C%20WormYOLO%20outperformed,%28mAP%200.5%3A0.95%29.%20In%20addition). This suggests that out-of-the-box, Deep-Worm-Tracker will **struggle to resolve the worm’s shape** during full coils or self-contact events (though it will keep detecting the blob). Fine-tuning the YOLO detector won’t help with pose ambiguity, since the issue is not detecting the worm but interpreting its shape. To handle self-overlap, you’d have to add custom logic: for instance, analyzing the segmented mask to infer which part of the worm is which. Deep-Worm-Tracker doesn’t do that internally. **In summary:** Deep-Worm-Tracker can track the worm through self-overlap (you won’t lose the worm’s position or ID), but **it cannot by itself unravel a coiled shape into the correct ordered skeleton**. With fine-tuning (training on images of coiled *E. fetida* worms), you might improve the segmentation quality of the worm vs. background, but you’d still need a strategy to get the correct pose when parts of the worm overlap. It might mis-identify the worm’s shape at that moment (e.g. the skeleton might loop or fragment). Consider Deep-Worm-Tracker a reliable **locator** during these events, but not a perfect **pose estimator** for them.
* **WormPose:** WormPose was specifically created to handle scenarios of **self-occlusion, coiling, and entangled postures** in single worms. It uses a CNN that was trained on synthetic images of worms in all sorts of poses, including extreme coils[[4]](https://journals.plos.org/ploscompbiol/article?id=10.1371/journal.pcbi.1008914#:~:text=animals%20such%20as%20the%20nematode,food). In tests on *C. elegans*, WormPose was able to reconstruct the full centerline through Omega turns and deep coils that stymied traditional methods[[11]](https://journals.plos.org/ploscompbiol/article?id=10.1371/journal.pcbi.1008914#:~:text=worm%20pose%20encoded%20as%20the,brightness%2C%20blur%2C%20noise%2C%20and%20occlusion). The model effectively “imagines” the worm’s hidden shape based on learned patterns. One important aspect: WormPose doesn’t just output a skeleton arbitrarily – it was designed to output an **ordered centerline from head to tail**, even for coiled shapes. Internally, if the worm is folded such that the head and tail swap their visible positions, WormPose will still attempt to place them correctly in the output sequence. (The pipeline uses past frame information and a head-tail inference step to keep this consistent.) In practice, **WormPose can handle self-overlap and omega bends without manual intervention**. Even if the worm ties itself in a knot, the trained CNN will predict a plausible skeleton. The only caveat is that WormPose needs to be *trained* for your specific scenario. With *C. elegans*, they achieved this by generating synthetic training data that included many coils and then training the CNN. You would do the same with *E. fetida* – once trained, there’s no further fine-tuning needed per video. It will generalize to any similar videos. WormPose also addresses the ambiguity of self-overlaps by effectively considering two possible skeleton interpretations (imagine a coil could be traversed in two ways, head-to-tail or tail-to-head) and choosing the one that fits the image best[[12]](https://journals.plos.org/ploscompbiol/article?id=10.1371/journal.pcbi.1008914#:~:text=measure%20between%20the%20input%20image,the%20bounding%20box%20of%20the). This is done automatically by the algorithm (it generates two synthetic images corresponding to the two mirror-image skeleton hypotheses and checks which matches the video frame better) – so it **resolves the “which side is on top” question for you**. In short, **WormPose is explicitly designed for exactly the issues of self-overlap, coiling, and reversal**. With a properly trained model, it should be able to track your compost worm through full coils and omega arches. Fine-tuning in the context of WormPose essentially means training a new model on your data (since there is no existing Eisenia model). This is a bit of upfront work, but it squarely addresses point (2): yes, WormPose (with training) can handle self-overlaps and omega bends, whereas Deep-Worm-Tracker will likely need additional help to do so.
* **DeepTangleCrawl and others (FYI):** It’s worth noting for completeness that very recent advances have combined tracking with pose estimation to handle overlaps. For example, **DeepTangleCrawl (DTC)** (Brown *et al.*, 2025) extends a worm pose model to use *temporal context* (multiple video frames at once) in addition to image data[[13]](https://journals.plos.org/ploscompbiol/article?id=10.1371/journal.pcbi.1013345#:~:text=show%20that%20a%20recently%20developed,will%20improve%20future%20drug%20screens). By looking at a sequence of frames, it can figure out how a worm is moving through a coil or when two worms collide, thereby maintaining a continuous skeleton track. DTC was shown to produce far fewer tracking failures on self-intersecting worm postures than both classical trackers and Deep-Worm-Tracker[[14]](https://journals.plos.org/ploscompbiol/article?id=10.1371/journal.pcbi.1013345#:~:text=When%20worms%20perform%20sharp%20turns%2C,most%20characteristic%20so%20missing%20them)[[15]](https://journals.plos.org/ploscompbiol/article?id=10.1371/journal.pcbi.1013345#:~:text=interrupted%20by%20collisions%20between%20worms,with%20legacy%20trackers%20including%20low). It essentially eliminates the gaps in trajectories that Tierpsy or Deep-Worm-Tracker would have during coils. While DTC isn’t a plug-and-play tool yet (it’s an active research project and currently tied to the Tierpsy 2.0 codebase), it represents the direction these tools are heading. For your immediate purposes, knowing this is mostly to reinforce that **the community recognizes the coil/overlap problem** and is developing solutions. WormPose is one such solution for single worms, and DWT on its own is not. So if your analysis heavily depends on accurately capturing those omega bends or coiled postures, leaning towards WormPose (or a similar advanced pose model) is advisable.

## Post-Processing and Adaptations for Self-Contact Frames

Even with a good tracker, **post-processing can significantly improve the accuracy and consistency** of the results, especially in tricky frames. Here are key adaptations to consider:

* **Head-Tail Identification:** In a single-worm experiment, it’s often important to know which end of the worm is the head (for example, to measure omega turn direction or to analyze reversals). Many trackers, including deep ones, output an *unlabeled* skeleton – basically a centerline curve. During a self-overlap or 180° reversal, it’s easy for an algorithm to get “turned around.” For instance, a tracker might start labeling the tail as the head after an omega turn because the worm’s orientation flipped. WormPose’s pipeline addresses this by enforcing a **consistent head-tail orientation over time**[[16]](https://iteal.github.io/wormpose/pipeline.html#:~:text=The%20network%20predicts%20the%20videos,tail%20orientation). It uses the fact that you provided head/tail info for some frames (from Tierpsy or manual input) and then keeps the orientation stable except when there’s strong evidence to flip it. If you adopt WormPose, you will benefit from this built-in head-tail correction. If you use Deep-Worm-Tracker or another method, you might need to add your own logic. One simple approach is to track the worm’s motion: the end that is leading the motion after a coil is likely the head. Another approach is to use image features – for example, *Eisenia fetida* have a clitellum (a band) closer to their head end, or the head might be slightly narrower. These cues can help you determine orientation. You could also manually set the head in the first frame and then assume the worm’s motion continuity (except when a clear reversal happens). **Motion smoothing algorithms** (next point) combined with known head movement patterns can detect when a reversal occurs (the worm’s direction of travel switches – an omega turn typically entails a reversal of motion). At that point, you can flip the orientation in the data. In summary, be prepared to implement a head/tail check as a post-processing step for Deep-Worm-Tracker outputs. This may involve comparing the skeleton orientation in consecutive frames or identifying morphological head-tail differences. WormPose users get this mostly handled by the software’s post-processing step.
* **Skeleton Smoothing and Gap Filling:** Long videos (27 minutes at 30 Hz is ~48,000 frames) inevitably have some noisy frames or minor tracking errors. To ensure the analysis is robust, you’ll want to smooth the worm’s pose over time. WormPose provides a **“postprocess” command that interpolates across dropped frames and smooths the skeleton trajectories**[[17]](https://iteal.github.io/wormpose/pipeline.html#:~:text=,interpolation%20and%20smoothing). This means if the network wasn’t confident in a particular frame (or dropped it), you can fill that in by interpolation, and you can apply a smoothing spline to the worm’s centerline points over time to reduce jitter. You can implement a similar concept with Deep-Worm-Tracker results: for example, if you notice a one-frame zigzag in the skeleton (perhaps due to a momentary lighting change or the worm’s texture confusing the segmentation), you can smooth that out by averaging the skeleton coordinates with neighboring frames. One powerful technique is to treat each point along the worm’s body as a time-series and low-pass filter them (since real worm motion is continuous and cannot jitter infinitely fast). Even simpler, you could represent the worm’s posture in terms of curvature along its body (the “eigenworm” features) and smooth those over time, which automatically smooths the pose. **Filling gaps**: If Deep-Worm-Tracker completely fails to skeletonize a rare frame (say the worm tied itself into a knot that the skeletonization code didn’t output), you can interpolate the skeleton from the last known good frame to the next good frame. Because the worm moves relatively slowly, missing a frame or two can be bridged by interpolation without big errors. The key is to ensure there are no sudden discontinuities in the tracked trajectory or shape, which could throw off analyses of speed, bending frequency, etc. By applying these smoothing and gap-filling steps, you significantly improve the quality of your data for subsequent analysis, irrespective of which tracker produced the initial data.
* **Resolving Ambiguous Frames:** Self-contact frames can introduce ambiguity that even a deep model might handle imperfectly. For instance, if a worm fully coils such that it forms a closed loop, some algorithms might output a looped skeleton or have an arbitrary break in the loop. WormPose’s strategy, as mentioned, is to generate two hypotheses and pick the best fit[[12]](https://journals.plos.org/ploscompbiol/article?id=10.1371/journal.pcbi.1008914#:~:text=measure%20between%20the%20input%20image,the%20bounding%20box%20of%20the). If you find yourself needing to do something manually with Deep-Worm-Tracker output, one idea is to use the frames immediately *before and after* the coil. Often, the worm goes into a coil and then out of it, and the frames on either side are unambiguous. You can enforce that the skeleton during the coil connects appropriately to those entry/exit frames. This might mean manually flipping a segment of the skeleton if it appears reversed, or choosing which branch of a bifurcated skeleton is the head side. Another adaptation could be to use a **simple physics prior**: a worm can’t teleport. So if in frame N the head was on one side and in frame N+1 the tracker thinks it’s on the opposite side of a loop, that’s likely a mistake – you can correct that by comparing positions and opting for the solution that yields a smaller movement from frame N to N+1. This kind of temporal consistency check can be coded as a post-processing filter. It’s essentially a sanity check on the output to catch impossible jumps or flips. WormPose’s use of frame sequences and selection of the best skeleton is an automated way to do this[[12]](https://journals.plos.org/ploscompbiol/article?id=10.1371/journal.pcbi.1008914#:~:text=measure%20between%20the%20input%20image,the%20bounding%20box%20of%20the). If not using WormPose, you may implement a simplified version of that logic tailored to your setup.

In summary, **post-processing is highly recommended**. At a minimum, ensure head/tail labels are consistent (so you accurately capture when the worm truly reverses direction vs. when the tracker just got confused) and apply smoothing/interpolation so that your worm’s motion data is clean. These steps will help resolve the remaining errors in self-contact frames and yield an accurate, continuous representation of the worm’s behavior.

## Hybrid Pipelines and Combining Methods

If no single tool perfectly addresses all needs, you can absolutely **combine approaches** to leverage their strengths. In fact, the design of WormPose itself is hybrid: it assumes you have used a classic tracker (like Tierpsy) to get easy frames done, and it only tackles the hard frames with the CNN[[18]](https://journals.plos.org/ploscompbiol/article?id=10.1371/journal.pcbi.1008914#:~:text=Our%20focus%20is%20on%20resolving,tail). Here are some ideas for a hybrid pipeline with your constraints:

* **Classical + Deep (Tierpsy + WormPose):** You could run a **classical tracker** (e.g. Tierpsy Tracker or even a simple OpenCV background subtraction and skeletonization) on your videos first. Given your conditions (single worm, clear background), a basic segmentation will work for the majority of frames where the worm is not self-overlapping. This will give you initial skeletons for, say, 90% of the video (and those are quick to obtain). You then feed these results into WormPose. WormPose will use those known-good skeletons as partial training data and *synthesize* many examples of coils and overlaps, effectively learning to predict the poses for the remaining 10% of frames that were problematic[[18]](https://journals.plos.org/ploscompbiol/article?id=10.1371/journal.pcbi.1008914#:~:text=Our%20focus%20is%20on%20resolving,tail). After training, WormPose will **re-process the video and fill in those gaps**, giving you a complete trajectory with coils resolved. This approach is efficient because you only spend heavy computation on the hard cases. It’s exactly how WormPose is intended to be used, and it plays nicely with your scenario (single worm on uniform background)[[19]](https://journals.plos.org/ploscompbiol/article?id=10.1371/journal.pcbi.1008914#:~:text=Our%20focus%20is%20on%20resolving,that%20the%20labeled%20frames%20provide)*. (*Note: Tierpsy can output the head-tail orientation and has its own smoothing; those provide a good starting point that WormPose then refines.) The outcome is a hybrid solution where classical tracking handles the straightforward frames (fast and no learning needed) and the deep model handles the tough frames (coils, omega turns) that classical methods miss[[9]](https://journals.plos.org/ploscompbiol/article?id=10.1371/journal.pcbi.1013345#:~:text=When%20worms%20perform%20sharp%20turns%2C,most%20characteristic%20so%20missing%20them). Many researchers favor this kind of pipeline to maximize both accuracy and efficiency.
* **Deep tracker + custom refinement:** Another combination is to use **Deep-Worm-Tracker for what it’s best at (real-time detection and robust tracking)**, and then add a custom step to fix the poses in overlap frames. For example, let Deep-Worm-Tracker run through the video and output bounding boxes, masks, and skeletons. Mark or flag the frames where you suspect the skeleton is wrong – this could be done by simple heuristics like “flag any frame where the worm’s area suddenly doubles (possible coil) or the skeleton has a sudden kink or branch.” Once you have those flagged frames, you can apply additional image processing or even a secondary neural network to them. One idea is to crop the worm from those frames (using DWT’s bounding box) and run a specialized **pose estimator on the cropped image**. This could be a lightweight CNN you train on a small set of coil images, or even running WormPose’s prediction step if you trained WormPose separately. In essence, Deep-Worm-Tracker ensures the worm is located and isolated (taking care of background noise, lighting variation, etc.), and then another module deals with deciphering the worm’s pose in the tricky situation. This two-stage approach (detect->then pose refine) is common in multi-animal tracking and could work here. If you don’t want to train another CNN, you could attempt a rule-based refinement: for instance, use morphological operations on the flagged frame’s worm mask to **split the worm at the self-contact point**. If a worm is in a full loop, the binary mask might look like a doughnut – you could detect the doughnut shape and break it at one point (e.g., erode the mask until it splits into two pieces, or find the narrowest point of contact and sever there). Then you skeletonize those two pieces separately and join them end-to-end. This would effectively reconstruct an ordered skeleton through a loop. Such a heuristic might work for simple overlaps (where the head just touches the body), and it can be implemented with image analysis (distance transform, etc.). It’s a bit of a custom engineering solution, but it can complement Deep-Worm-Tracker’s output if you find that only a small fraction of frames need this treatment.
* **Pose estimation frameworks:** If you are open to training with manual labels, you could also incorporate tools like **DeepLabCut or SLEAP** in a hybrid way. For example, use Deep-Worm-Tracker to get the worm’s bounding box trajectory, then take a subset of frames (especially those with coils) and manually label a series of points along the worm’s body. Train a DLC/SLEAP model on those points – this model will effectively be a pose estimator that knows how to output a skeleton for your worm. Then run this model on the video (with the worm cropped/aligned by the detection). The advantage is that DLC/SLEAP are highly optimized and can track keypoints quickly, and by training on your worm you ensure it knows *Eisenia*-specific features. This is arguably overkill given WormPose can achieve similar results without manual labeling, but I mention it as an alternative path. One might choose this if, say, they already have DLC set up or if they want to track specific landmarks on the worm (head, tail, midbody bends) rather than a full continuous skeleton. It’s a hybrid approach in the sense that one model (DWT) handles detection/tracking and another handles pose. However, keep in mind the development time: labeling and training a keypoint model is a non-trivial amount of work.

Overall, combining methods can be very effective. **WormPose + a basic tracker is a proven hybrid combo**[[18]](https://journals.plos.org/ploscompbiol/article?id=10.1371/journal.pcbi.1008914#:~:text=Our%20focus%20is%20on%20resolving,tail). Deep-Worm-Tracker + some custom overlap-resolution code is another viable combo. The specific choice may depend on your comfort with training models versus writing image processing code. The end goal is to ensure that no important behavioral event (like an omega turn) is missed due to tracking failure. A hybrid pipeline gives you a safety net by using one approach to cover the weaknesses of another.

## Recommendations for Your Use-Case

Considering your setup and needs, here are **clear recommendations**:

1. **For the highest accuracy in challenging poses (coils/omegas)**: Implement a WormPose-based solution. This will require some initial effort – you’ll use a classical tracker (or even Deep-Worm-Tracker’s output or OpenCV) to get skeletons for non-coiled frames, then train WormPose on your *E. fetida* videos. The payoff is a model that can **reliably recover the worm’s full posture even during self-overlap**[[11]](https://journals.plos.org/ploscompbiol/article?id=10.1371/journal.pcbi.1008914#:~:text=worm%20pose%20encoded%20as%20the,brightness%2C%20blur%2C%20noise%2C%20and%20occlusion). WormPose will handle head-tail orientation consistency and provide smoothed, gap-free skeleton data as part of its pipeline[[16]](https://iteal.github.io/wormpose/pipeline.html#:~:text=The%20network%20predicts%20the%20videos,tail%20orientation)[[17]](https://iteal.github.io/wormpose/pipeline.html#:~:text=,interpolation%20and%20smoothing). This approach directly addresses the core of your question: WormPose is specifically demonstrated to handle “self-occluded, coiled shapes”[[4]](https://journals.plos.org/ploscompbiol/article?id=10.1371/journal.pcbi.1008914#:~:text=animals%20such%20as%20the%20nematode,food), which includes omega bends and full coiling. Given your single-worm, high-resolution videos, WormPose is well-suited to produce accurate posture data throughout the 27-minute recordings. The processing time for WormPose involves the training phase (which could take some hours on a GPU, depending on how much data you use), but once trained, **inference on 50k frames is quite reasonable** (potentially on the order of a few minutes). This is within your limit, since you mentioned being open to deeper integration if time per video is manageable. The benefit is that you won’t need elaborate post-hoc fixes for coils – the model should get them right, and you can trust the output for downstream analysis of behaviors like omega turns or coiling frequencies.
2. **If you need a quicker, out-of-the-box solution**: You can start with Deep-Worm-Tracker, with some caveats. It will give you immediate tracking and basic pose outputs without a lengthy training process. To make it work for *E. fetida*, you might do the following: fine-tune the YOLO detector on a small set of your worm images (ensuring the larger worm is learned – this could be as simple as adding 50 labeled frames and training for a few minutes). Then run the tracker on your videos. It will output a skeleton for each frame. You should then **inspect the output, specifically at times when the worm coils or reverses**. Likely, you will see some errors there (e.g. the skeleton might loop around or the head marker might jump). Plan to apply post-processing: for example, use a smoothing spline through the skeleton coordinates to iron out any one-frame anomalies, and implement a head-tail orientation check as discussed. Deep-Worm-Tracker’s speed is a big plus – it can process 30 FPS in real-time (or faster) on a GPU[[3]](https://github.com/knaticat/Deep-Worm-Tracker#:~:text=change%20in%20magnification%2C%20dust%20particles,worm%20trajectories%20are%20also%20highlighted), so your 27-minute video can be analyzed in well under 27 minutes of compute. This means you can iterate quickly. If you find that certain coiling events consistently trip it up, you could then decide to incorporate a secondary fix (maybe even use a few WormPose predictions just for those frames, if training a whole model is too much – there is a way to run WormPose in a **calibration mode** on single frames if you supply partial data[[16]](https://iteal.github.io/wormpose/pipeline.html#:~:text=The%20network%20predicts%20the%20videos,tail%20orientation)). Essentially, **Deep-Worm-Tracker can serve as the backbone of a pipeline**, but you’ll likely need to build some custom logic around it to handle the “worst-case” poses. This is a valid approach if your priority is to avoid extensive model training and you’re comfortable writing some image analysis or filtering code to patch the remaining issues.
3. **Hybrid approach for best of both**: You don’t necessarily have to choose one exclusively. For instance, you could use Deep-Worm-Tracker to get the initial track and identify frames of interest (coils, etc.), then apply WormPose just to those segments. Or run Tierpsy for basics and WormPose for refinement. Given your description, a **reasonable plan is**: run a quick segmentation-based tracker on the video to get an initial skeleton and identify frames where it fails (likely the overlapping frames). Then train WormPose with that data – this training process effectively gives you a model that knows how to handle the failed frames. Finally, run WormPose to get the final, clean pose data for the entire video. This way, you leverage simplicity (segmentation) for the easy parts and ML for the hard parts. It’s how the WormPose authors tackled the problem and is well-suited to single worms on uniform backgrounds[[18]](https://journals.plos.org/ploscompbiol/article?id=10.1371/journal.pcbi.1008914#:~:text=Our%20focus%20is%20on%20resolving,tail). The result will be highly accurate and require minimal manual intervention during analysis.
4. **Plan for post-processing**: Whichever route you take, incorporate the post-processing steps we discussed. Ensure head/tail labels are consistent (especially if you care about distinguishing forward vs reverse movement). Use smoothing to remove jitter – this will greatly improve any metrics you derive (like bending amplitude, omega turn duration, etc., since you won’t be thrown off by noise). The post-processing is not too onerous – often it can be done with a small script after you get the tracker output (e.g., reading the skeleton coordinates and applying a filter). WormPose’s built-in postprocessing might already suffice[[17]](https://iteal.github.io/wormpose/pipeline.html#:~:text=,interpolation%20and%20smoothing), and for Deep-Worm-Tracker output you can use a similar approach.

**Recommendation Summary:** If your analysis depends critically on correctly capturing self-overlap events and you want confidence in those segments, invest the time to use **WormPose (with your own training)** – it will handle *Eisenia fetida* just fine and is explicitly made for coils and omega bends[[11]](https://journals.plos.org/ploscompbiol/article?id=10.1371/journal.pcbi.1008914#:~:text=worm%20pose%20encoded%20as%20the,brightness%2C%20blur%2C%20noise%2C%20and%20occlusion). If instead you need a quicker solution and are okay doing some custom fixes, **Deep-Worm-Tracker can be employed**, keeping in mind its pose outputs around overlaps will need verification. In many cases, a **hybrid pipeline is the safest bet**: use fast methods to get started and deep learning models to handle the tough cases. This way, you ensure that even when your worm ties itself in a knot or performs a reversal, you have the tools in place to accurately track it throughout. By following these strategies, you can confidently analyze single *Eisenia fetida* worm videos for complex behaviors, leveraging deep learning where it’s most effective and simpler techniques where they suffice. Good luck with your worm tracking – with the above approach, you should be able to capture those omega arches and coils with high fidelity, enabling the behavioral insights you’re after.

**Sources:**

* Hebert *et al.*, “WormPose: Image synthesis and convolutional networks for pose estimation in *C. elegans*,” *PLoS Comput. Biol.* 17(4): e1008914 (2021) – WormPose handles single-worm coiled postures[[4]](https://journals.plos.org/ploscompbiol/article?id=10.1371/journal.pcbi.1008914#:~:text=animals%20such%20as%20the%20nematode,food)[[5]](https://journals.plos.org/ploscompbiol/article?id=10.1371/journal.pcbi.1008914#:~:text=misidentified%20as%20an%20increase%20in,bodied%20and%20deformable%20organisms).
* Banerjee *et al.*, “Deep-Worm-Tracker: Deep learning methods for accurate detection and tracking for behavioral studies in *C. elegans*,” *Appl. Anim. Behav. Sci.* 266: 106024 (2023) – YOLO/SORT tracker for worms (pretrained on *C. elegans*)[[2]](https://www.nature.com/articles/s41598-025-93533-0?error=cookies_not_supported&code=45077ede-dc61-41c4-9bd3-b1a52daebcaf#:~:text=tracking%20methods%20has%20also%20emerged,training%20times%2C%20and%20slow%20inference).
* Zhang *et al.*, “A high precision method of segmenting complex postures in *C. elegans* and deep phenotyping to analyze lifespan,” *Sci. Rep.* 13: 11021 (2023) – Discusses Deep-Worm-Tracker’s performance and overlap limitations[[10]](https://www.nature.com/articles/s41598-025-93533-0?error=cookies_not_supported&code=45077ede-dc61-41c4-9bd3-b1a52daebcaf#:~:text=On%20three%20datasets%2C%20WormYOLO%20outperformed,%28mAP%200.5%3A0.95%29.%20In%20addition).
* Weheliye *et al.*, “A neural network model enables worm tracking in challenging conditions…” *PLoS Comput. Biol.* 21(8): e1013345 (2025) – Introduces DeepTangleCrawl, showing legacy trackers fail on self-intersections[[14]](https://journals.plos.org/ploscompbiol/article?id=10.1371/journal.pcbi.1013345#:~:text=When%20worms%20perform%20sharp%20turns%2C,most%20characteristic%20so%20missing%20them) and achieving gap-free tracking through coils[[15]](https://journals.plos.org/ploscompbiol/article?id=10.1371/journal.pcbi.1013345#:~:text=interrupted%20by%20collisions%20between%20worms,with%20legacy%20trackers%20including%20low).
* *Deep-Worm-Tracker* GitHub README – Notes on dataset (3000 images) and skeletonization feature[[1]](https://github.com/knaticat/Deep-Worm-Tracker#:~:text=,the%20YOLO%20object%20detection%20model)[[3]](https://github.com/knaticat/Deep-Worm-Tracker#:~:text=change%20in%20magnification%2C%20dust%20particles,worm%20trajectories%20are%20also%20highlighted).
* WormPose Documentation – Pipeline requiring Tierpsy input and post-processing (head-tail orientation, smoothing)[[18]](https://journals.plos.org/ploscompbiol/article?id=10.1371/journal.pcbi.1008914#:~:text=Our%20focus%20is%20on%20resolving,tail)[[16]](https://iteal.github.io/wormpose/pipeline.html#:~:text=The%20network%20predicts%20the%20videos,tail%20orientation).

[[1]](https://github.com/knaticat/Deep-Worm-Tracker#:~:text=,the%20YOLO%20object%20detection%20model) [[3]](https://github.com/knaticat/Deep-Worm-Tracker#:~:text=change%20in%20magnification%2C%20dust%20particles,worm%20trajectories%20are%20also%20highlighted) GitHub - knaticat/Deep-Worm-Tracker: Realtime C. elegans worm tracker based on Deep Object Detection (YOLO) and Tracking (Strong SORT) models

<https://github.com/knaticat/Deep-Worm-Tracker>

[[2]](https://www.nature.com/articles/s41598-025-93533-0?error=cookies_not_supported&code=45077ede-dc61-41c4-9bd3-b1a52daebcaf#:~:text=tracking%20methods%20has%20also%20emerged,training%20times%2C%20and%20slow%20inference) [[10]](https://www.nature.com/articles/s41598-025-93533-0?error=cookies_not_supported&code=45077ede-dc61-41c4-9bd3-b1a52daebcaf#:~:text=On%20three%20datasets%2C%20WormYOLO%20outperformed,%28mAP%200.5%3A0.95%29.%20In%20addition) A high precision method of segmenting complex postures in Caenorhabditis elegans and deep phenotyping to analyze lifespan | Scientific Reports

<https://www.nature.com/articles/s41598-025-93533-0?error=cookies_not_supported&code=45077ede-dc61-41c4-9bd3-b1a52daebcaf>

[[4]](https://journals.plos.org/ploscompbiol/article?id=10.1371/journal.pcbi.1008914#:~:text=animals%20such%20as%20the%20nematode,food) [[5]](https://journals.plos.org/ploscompbiol/article?id=10.1371/journal.pcbi.1008914#:~:text=misidentified%20as%20an%20increase%20in,bodied%20and%20deformable%20organisms) [[6]](https://journals.plos.org/ploscompbiol/article?id=10.1371/journal.pcbi.1008914#:~:text=protrusions%20are%20still%20visible%2C%20especially,the%20desired%20worm%20width%2C%20complete) [[7]](https://journals.plos.org/ploscompbiol/article?id=10.1371/journal.pcbi.1008914#:~:text=An%20important%20model%20system%20for,synthetic%20data%20as%20well%20as) [[11]](https://journals.plos.org/ploscompbiol/article?id=10.1371/journal.pcbi.1008914#:~:text=worm%20pose%20encoded%20as%20the,brightness%2C%20blur%2C%20noise%2C%20and%20occlusion) [[12]](https://journals.plos.org/ploscompbiol/article?id=10.1371/journal.pcbi.1008914#:~:text=measure%20between%20the%20input%20image,the%20bounding%20box%20of%20the) [[18]](https://journals.plos.org/ploscompbiol/article?id=10.1371/journal.pcbi.1008914#:~:text=Our%20focus%20is%20on%20resolving,tail) [[19]](https://journals.plos.org/ploscompbiol/article?id=10.1371/journal.pcbi.1008914#:~:text=Our%20focus%20is%20on%20resolving,that%20the%20labeled%20frames%20provide) WormPose: Image synthesis and convolutional networks for pose estimation in C. elegans | PLOS Computational Biology

<https://journals.plos.org/ploscompbiol/article?id=10.1371/journal.pcbi.1008914>

[[8]](https://journals.plos.org/ploscompbiol/article?id=10.1371/journal.pcbi.1013345#:~:text=have%20addressed%20these%20issues%20to,screens%2C%20even%20for%20data%20that) [[9]](https://journals.plos.org/ploscompbiol/article?id=10.1371/journal.pcbi.1013345#:~:text=When%20worms%20perform%20sharp%20turns%2C,most%20characteristic%20so%20missing%20them) [[13]](https://journals.plos.org/ploscompbiol/article?id=10.1371/journal.pcbi.1013345#:~:text=show%20that%20a%20recently%20developed,will%20improve%20future%20drug%20screens) [[14]](https://journals.plos.org/ploscompbiol/article?id=10.1371/journal.pcbi.1013345#:~:text=When%20worms%20perform%20sharp%20turns%2C,most%20characteristic%20so%20missing%20them) [[15]](https://journals.plos.org/ploscompbiol/article?id=10.1371/journal.pcbi.1013345#:~:text=interrupted%20by%20collisions%20between%20worms,with%20legacy%20trackers%20including%20low) A neural network model enables worm tracking in challenging conditions and increases signal-to-noise ratio in phenotypic screens | PLOS Computational Biology

<https://journals.plos.org/ploscompbiol/article?id=10.1371/journal.pcbi.1013345>

[[16]](https://iteal.github.io/wormpose/pipeline.html#:~:text=The%20network%20predicts%20the%20videos,tail%20orientation) [[17]](https://iteal.github.io/wormpose/pipeline.html#:~:text=,interpolation%20and%20smoothing) WormPose pipeline — WormPose documentation

<https://iteal.github.io/wormpose/pipeline.html>