CRAFT: CODING AGENTS FOR VISUAL TRACKING CONTROLLER DESIGN

Anonymous authors
Paper under double-blind review

ABSTRACT

A visual tracker couples frozen visual model components with an online tracking controller: ordinary program logic that maintains state and determines how visual components are used over time. Although visual models have advanced substantially, controllers still encode manually designed choices about where to search, how to estimate target state, which templates to retain, and when to recover. We introduce Code Refinement by Agents for Tracking (**CRAFT**), a computational framework in which coding agents propose and test controller revisions offline under purpose-constrained design tasks while keeping the visual model components and their internal inference procedures fixed. CRAFT organizes exploration around four controller operations: search-region selection, response decoding, template-memory management, and failure recovery. First, candidate revisions are sampled for each operation and selected according to tracking performance on a development split. A composing agent then integrates the selected mechanisms without access to their performance metrics. Guided by evidence from earlier revisions, joint refinement then returns to the coupled controller space to address interactions among the discovered mechanisms. The resulting controller runs as ordinary tracker code, with no language model at inference. Using the TNL2K training split as its sole source of tracking feedback, CRAFT improves performance on every reported held-out benchmark: LaSOT AUC increases from 72.6% to 74.6% for a base-scale one-stream tracker and from 77.6% to 78.8% for its SAM3-augmented counterpart, with gains of 2.1 to 3.0 points on $\text{LaSOT}_{\text{ext}}$ and the TNL2K test split. Code, discovered controllers, and models will be released.

1 INTRODUCTION

Visual object tracking aims to localize an arbitrary target throughout a video given its location in the first frame (Lucas & Kanade, 1981; Wu et al., 2013). Since its early formulations, a tracker has combined an *appearance model*, which matches the target, with online decision logic that governs how the model is used over time (Kalal et al., 2012; Hong et al., 2015; Ma et al., 2015). Modern trackers have turned the appearance model into a learned, template-conditioned visual model: a Siamese network (Bertinetto et al., 2016; Li et al., 2018), a Transformer with template-search fusion (Chen et al., 2021b; Yan et al., 2021; Cui et al., 2022), or a one-stream encoder that processes templates and the search region jointly (Ye et al., 2022; Lin et al., 2024; Hong et al., 2024).

The learned visual model, however, is not the entire tracker. Given templates and a search crop, the visual model produces a response; a surrounding online program decides where the next crop is, how responses become box predictions, which observations enter the template memory, and how the tracker recovers after target loss. We call this online program the **tracking controller** (see Figure 1).

The visual model and the tracking controller have advanced at different speeds. The model is optimized by gradient descent on large video datasets, and much recent tracking progress has come from stronger pretrained backbones (Lin et al., 2022; Ye et al., 2022; Lin et al., 2024). The controller, by contrast, is still mostly hand-written program logic inherited from the Siamese era (Bertinetto et al., 2016; Li et al., 2018): it crops the search region at a fixed scale, favors nearby positions when decoding, updates templates at fixed intervals, and occasionally triggers re-detection (Chen et al., 2021b; Yan et al., 2021; Cui et al., 2022). The slower progress of the controller has a practical cause. Controller design is discrete, stateful, and coupled across frames, so it is not trained directly

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

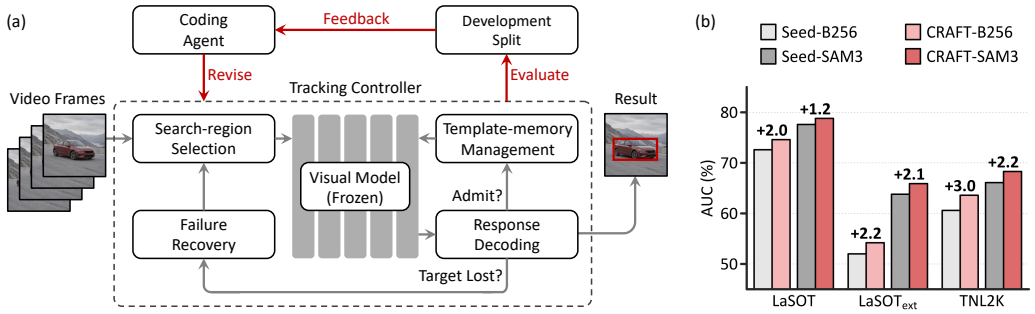


Figure 1: **The tracking controller and its agentic design.** (a) A visual tracker couples a visual model with a tracking controller, the online program that governs how the model is used over time; grey arrows show the per-frame data flow and red arrows the offline design loop of CRAFT, in which a coding agent revises the controller from development-split feedback with the visual model frozen. (b) Held-out AUC of a base-scale tracker (B256) and its SAM3-augmented counterpart (SAM3) with the seed and with the CRAFT controller; numbers are gains over the seed.

from data. Each new mechanism requires manual formulation and evaluation through full tracking runs. As a result, controller exploration remains limited by human design effort and experimental iteration, and is thus studied less systematically than the visual model it surrounds.

Coding agents offer a way to reduce this manual design burden. A coding agent is a tool-using language-model system that reads a codebase, forms a hypothesis about a failure, implements and tests a change, and interprets the outcome (Chen et al., 2021a; Jimenez et al., 2024). These are the same steps that make manual controller design slow. This raises a question: can the controller be optimized from tracking feedback by coding agents, while fixing the visual model?

Here, we introduce Code Refinement by Agents for Tracking (**CRAFT**), a computational framework that structures agent-driven controller design over a frozen visual model in three stages. In **scoped exploration**, coding agents start from the same seed controller and run independent exploration processes, one per purpose-constrained design task. Each process proposes, implements, and tests revisions offline, and the best revision from each task is selected according to development-split performance. In **metric-blind composition**, a fresh composing agent integrates the selected code changes, with their performance metrics hidden, into one controller. In **joint refinement**, the agent receives the composed program and the complete record of the scoped explorations, and then may further revise mechanisms to resolve interactions that appear only after composition. Throughout, the visual model stays fixed. The resulting output is an optimized online tracking program with no language model at inference.

We evaluate CRAFT on two base trackers of different strength. The first is a base-scale one-stream Transformer tracker (Ye et al., 2022), the simplest form of template matching in current use, whose seed controller makes the most common choices in this family. The second asks whether the framework can still improve a much stronger starting point: the same one-stream tracker augmented with SAM3 video propagation (Carion et al., 2026), which improves accuracy but runs below real time. Using only the TNL2K training split (Wang et al., 2021) for development and with the visual models untouched, CRAFT improves every held-out benchmark on both trackers, raising LaSOT (Fan et al., 2019) AUC from 72.6% to 74.6% on the first and from 77.6% to 78.8% on the second, and LaSOT$_{\text{ext}}$ (Fan et al., 2021) and TNL2K AUC by 2.1 to 3.0 points. In both cases a single controller, with the same program (and the same hyperparameters), is used on every benchmark, and its gains carry over to the short-term benchmarks (Müller et al., 2018; Huang et al., 2021).

In summary, this paper makes three contributions:

- We identify the tracking controller as an underexplored source of tracking performance and formulate its design as program optimization around a frozen visual model.
- We introduce CRAFT, a coding-agent framework that decomposes controller design into four purpose-constrained design tasks and searches them through scoped exploration, metric-blind composition, and joint refinement.
- We show that CRAFT produces readable, deployable controllers that improve two base trackers across held-out benchmarks, rather than merely tuning hyperparameters.

2 Related Work

In this section, we position CRAFT at the intersection of two research lines: controller design in visual tracking and language-model agents for program design.

2.1 Controller Design in Visual Object Tracking

Tracking controllers organize four major online operations around the visual model: selecting the search region, decoding the response, managing template memory, and recovering from failure. Each operation has developed its own hand-designed mechanisms. For *search-region selection*, most trackers crop a fixed multiple of the previous box (Bertinetto et al., 2016; Ye et al., 2022), while long-term trackers either expand the search after failure (Yan et al., 2019; Dai et al., 2020) or search the full image throughout (Huang et al., 2020). For *response decoding*, cosine windows and scale penalties bias the selected response toward the previous state (Li et al., 2018; Ye et al., 2022), whereas KeepTrack (Mayer et al., 2021) links candidate peaks across frames to reduce distractor switches. For *template-memory management*, trackers refresh templates from confident predictions at fixed intervals (Yan et al., 2021; Cui et al., 2022), or update a discriminative target model online (Danelljan et al., 2019; Bhat et al., 2019; Mayer et al., 2022). For *failure recovery*, long-term trackers combine a local tracker with a verifier or meta-updater that decides when to distrust the local prediction and when to re-detect (Yan et al., 2019; Dai et al., 2020). Recent trackers built on SAM 2 (Ravi et al., 2025; Videnovic et al., 2025; Ding et al., 2025; Yang et al., 2026) make the role of the controller especially clear: long-term performance can be improved through hand-designed motion and memory logic around a foundation segmentation model. These studies demonstrate that controller logic matters, but they also illustrate the prevailing design pattern: mechanisms are conceived, implemented, and validated by hand, usually for one operation at a time.

A related set of methods learns one controller operation instead of writing it entirely by hand. UpdateNet (Zhang et al., 2019) learns a template-update function, while reinforcement learning has been used to decide when to update the template (Zhao et al., 2021), which tracker to run at each frame (Song et al., 2020), and how to adjust tracker hyperparameters online (Dong et al., 2021). These policies are trained through fixed interfaces for individual operations, leaving the remaining controller logic hand-written. More recently, agentic trackers call a vision-language model during inference to support individual operations, such as generating and updating the target description (Sun et al., 2024; Li et al., 2025) or verifying and re-detecting the target after loss (Shen et al., 2026; Huang et al., 2026). The comparison with CRAFT is therefore along four axes: the design target, the designer, the feedback used for evaluation, and the deployed artifact. Instead of learning or prompting one operation while the rest of the controller is fixed, CRAFT makes all four controller operations the object of joint automated design; instead of using a language model as part of the online tracker, it uses an offline coding agent to revise controller source code; instead of designing through fixed operation-level interfaces, it evaluates complete tracking runs; and instead of deploying an agentic system, it outputs a standard online tracking program.

2.2 Language-Model Agents for Program Design

Automated design predates language models, but earlier systems typically constrain the search space before optimization begins. Neural architecture search selects architectures from a fixed space by reinforcement learning (Zoph & Le, 2017) or by gradient descent on a continuous relaxation (Liu et al., 2019), and AutoML-Zero evolves learning algorithms from primitive operations (Real et al., 2020). Language models relax this constraint by proposing candidates directly as code (Lehman et al., 2024). FunSearch (Romera-Paredes et al., 2024) and AlphaEvolve (Novikov et al., 2025) evolve programs for mathematical and algorithmic problems under an automatic evaluator, Evolution of Heuristics (Liu et al., 2024) applies the same principle to combinatorial heuristics, and OPRO (Yang et al., 2024) prompts the model to propose new solutions from a trajectory of scored ones. Eureka (Ma et al., 2024) uses a coding model to write reward functions that are then used to train reinforcement learning policies. These systems share a common structure that is useful for CRAFT: candidate programs or specifications are repeatedly generated, evaluated, and selected using automatic feedback.

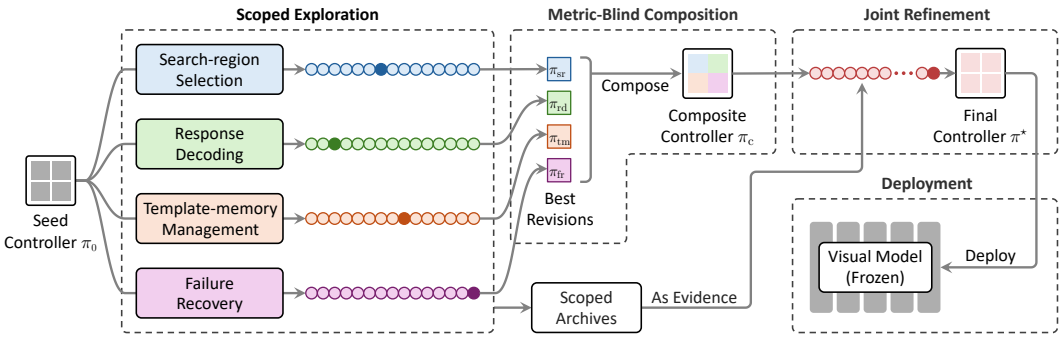


Figure 2: **Overview of CRAFT.** Each circle denotes one agent round: a controller revision evaluated once on the development split; filled circles mark the scoped revisions and final joint-refinement round selected by development-split AUC. Scoped exploration starts four independent campaigns from the seed controller π_0, one for each operation-defined design space, and returns the selected revisions π_{sr}, π_{rd}, π_{tm}, and π_{fr}, together with scoped archives containing the code and evidence from every scoped round. Metric-blind composition integrates the four selected revisions, without exposing their metrics to the composing agent, into a composite controller π_{c}. Joint refinement then revises π_{c} with the scoped archives available as evidence; the joint-refinement round with the highest development-split AUC becomes the final controller $\pi^\star$, which is deployed unchanged as ordinary tracker code around the frozen visual component or model set.

Coding agents move this generate–evaluate–revise loop inside entire codebases, reading, modifying, and testing repository code (Chen et al., 2021a; Jimenez et al., 2024). This makes it possible to automate larger engineering workflows. AIDE (Jiang et al., 2025) and the AI Scientist (Lu et al., 2026) use such agents to run machine learning engineering and research loops, while MLE-bench (Chan et al., 2025) evaluates their performance on competition tasks. CRAFT brings this paradigm to a causal online controller around a frozen perception model, using full tracking runs as feedback while still producing deployable tracker code.

3 Proposed Method: CRAFT

In this section, we formulate CRAFT as offline program search for the tracking controller while the visual model stays frozen. Starting from a seed controller π_0 and a development split $\mathcal{D}_{\mathrm{dev}}$, CRAFT treats the controller's four online operations as purpose-constrained design tasks, then uses scoped exploration, metric-blind composition, and joint refinement to produce a causal controller program.

3.1 Tracking Controller

Let $f_{\boldsymbol{\theta}}$ denote a frozen template-conditioned visual model. Given template crops $\mathbf{Z}$ and a search crop $\boldsymbol{x}$, it returns a response map $\hat{\mathbf{R}}$ over the search locations and a box regression $\hat{\mathbf{B}}$ at each location,

$$\left(\hat{\mathbf{R}},\, \hat{\mathbf{B}}\right) = f_{\boldsymbol{\theta}}(\mathbf{Z}, \boldsymbol{x})\,. \tag{1}$$

This equation describes only the visual computation. The model holds no state across frames; the tracking controller is the online program that decides how the model is called, how its outputs are interpreted, and how information is carried forward.

A visual tracker is the pair $(f_{\boldsymbol{\theta}}, \pi)$. The first frame initializes the controller state $\boldsymbol{s}_1$ from the image and annotated box $(\mathbf{I}^{(1)}, \boldsymbol{b}^{(1)})$. For each later frame $t \geq 2$, the controller π has the causal signature:

$$\boldsymbol{s}_t = \pi(\mathbf{I}^{(t)}, \boldsymbol{s}_{t-1}; f_{\boldsymbol{\theta}}),$$

where the semicolon denotes access to the frozen visual model. The state stores recent box predictions and associated confidences, motion estimates, and template memory. Within this call, the tracking controller comprises four major operations:

- *Search-region selection*: choosing the search crop $\boldsymbol{x}^{(t)}$ from $\mathbf{I}^{(t)}$, including its center, extent, and any rule-based adjustment from recent motion.

- *Response decoding*: running the controller code that turns $(\hat{\mathbf{R}}^{(t)}, \hat{\mathbf{B}}^{(t)})$ into the returned box $\hat{\boldsymbol{b}}^{(t)}$ and confidence $c^{(t)}$, including programmed spatial-prior weights, candidate selection or aggregation rules, and acceptance tests.
- *Template-memory management*: deciding which observations enter memory, which are removed, and which form the template tuple $\mathbf{Z}^{(t+1)}$ used on the next frame.
- *Failure recovery*: detecting suspected loss, searching beyond the normal region, and deciding when a candidate is reliable enough to resume normal tracking.

For each operation, its design space is the set of admissible source-code revisions whose primary causal purpose is to change that operation while preserving the seed behavior of the other operations. These spaces are fixed before a run and are defined by mechanism purpose rather than by a file or function partition; the definitions given to the agents are listed in Appendix A. They are not independent at inference time. A wider crop changes the peaks available to the decoder, an admitted template changes later responses, and a recovery rule changes which observations are allowed to update memory.

This coupling motivates the design procedure in Figure 2. A development-split evaluation is a full tracking run, so each evaluation is expensive and its score reflects all four operations together. If one revision changes several operations, the feedback is difficult to attribute; if the other three operations are held at the seed, the same evaluation is clearer evidence about the active operation. CRAFT therefore begins with scoped exploration, where each round is constrained to one operation-defined design space. Scoped evidence alone is not enough, because a mechanism that helps in isolation may interact poorly once the other operations also change. CRAFT therefore composes the scoped results and then refines the composed controller jointly, using the scoped archives as evidence about what each operation contributed.

Seed controller. The seed π_0 is a standard and robust controller for recent one-stream trackers (Chen et al., 2025; Feng et al., 2025): it crops the search region at a fixed factor around the previous box, multiplies the response by a cosine window, decodes the box at the maximum, and updates the templates at a fixed interval when the confidence exceeds a threshold; it has no failure-recovery mechanism. With it, the seed tracker is already competitive with the state of the art (Table 1). Starting every scoped campaign from the same π_0 makes later controller changes attributable to the design procedure rather than to a changed visual model or starting point.

Design objective. Given a development split $\mathcal{D}_{\text{dev}}$, CRAFT uses development-split AUC as a black-box objective over admissible controller programs,

$$\pi^\star = \arg\max_{\pi \in \Pi} \ \mathrm{AUC}(f_{\boldsymbol{\theta}}, \pi; \mathcal{D}_{\text{dev}}) \,, \tag{2}$$

where Π is the full set of programs satisfying four constraints: (i) the visual model is fixed, both its weights $\boldsymbol{\theta}$ and its inference procedure; (ii) π is causal at inference and may use only the initialization box, the current and past frames, and its own state, with no ground truth after initialization and no language model or network service; (iii) π contains no rule keyed to a sequence, class, object name, or dataset identity; (iv) the held-out benchmarks $\mathcal{D}_{\text{held}}$ are invisible to the design procedure: no held-out data or results enter any agent's input or any selection, and the selected program is evaluated on them once. Constraints (i) to (iii) keep the result a deployable tracker; (iv) makes the held-out results a genuine test of transfer. The objective in Equation (2) is a black-box function of source code. CRAFT does not enumerate Π or claim to solve this objective globally; instead, it samples a finite set of admissible source-code revisions through the three-stage procedure below and applies the same development-split criterion to choose the final joint-refinement round.

3.2 Agent Round

CRAFT samples controller revisions in rounds. In one round, a coding agent proposes one coherent source-code revision, the revised tracker is evaluated once on $\mathcal{D}_{\text{dev}}$, and the result is appended to an archive. A campaign is a sequence of such rounds sharing a starting program and one archive.

The agent works directly in the tracker's source tree, where the controller occupies a small and well-delimited surface, namely the tracker file and its inference configuration. At the start of a round, the agent receives the campaign's starting program, the constraints of Section 3.1, and the complete archive of the campaign so far: for every earlier round, the source snapshot, aggregate development-

split metrics, per-sequence AUC deltas against the starting program, speed, and failure logs. It may inspect the videos, annotations, and predictions of $\mathcal{D}_{\mathrm{dev}}$ for failure analysis. The agent then chooses one parent, the starting program or any completed round in the campaign, implements one coherent revision of the controller on top of it, verifies with a metric-free smoke test that the program tracks a short frame prefix, and submits. CRAFT snapshots the source, runs exactly one full evaluation on $\mathcal{D}_{\mathrm{dev}}$, and appends the result to the archive; a crash or timeout consumes the round.

Two rules keep campaigns exploratory and auditable. First, no incumbent is imposed: since any earlier round may serve as a parent, a campaign can branch, revert, and recombine rather than follow a single greedy path. Second, each round runs in a fresh agent session, so nothing but the archived record carries over. The archive, with every round's transcript, diff, source, and metrics, records how each mechanism arose.

3.3 Design Procedure

The design procedure has three stages. Scoped exploration and joint refinement are campaigns built from the rounds above, and metric-blind composition is a single integration step between them.

Scoped exploration. Scoped exploration samples revisions within one design space at a time, so that every evaluation is evidence about a single operation. CRAFT runs four independent campaigns from the same seed π_0, one per operation-defined design space, each for K_1 rounds. In every round, the agent declares its parent and active space, and the revision must preserve the seed behavior of the three inactive operations except for shared state changes required by the active mechanism. The four campaigns share only the seed: no source tree, archive, or metric crosses between them, and they can run in parallel. Each archive is therefore evidence about one operation: which mechanisms were tried, what each did on which sequences, and how far the space can be pushed by itself. The best revision of each space, by development-split AUC with ties broken toward the earlier round, is passed to composition, and the four complete archives, the *scoped archives*, to joint refinement.

Metric-blind composition. Composition turns the selected revisions π_{sr}, π_{rd}, π_{tm}, and π_{fr} into one controller. They are complete source trees developed in isolation, and their changes may touch shared state or control flow, so composition is not simply patch stacking. A fresh composing agent receives the seed and the four selected trees, but not their development-split metrics, and integrates the behavior of all four mechanisms into a composite controller π_{c}. Hiding the metrics keeps composition an integration task: the composing agent is asked to preserve the selected mechanisms and resolve implementation conflicts, not to re-rank the scoped campaigns. The composite passes the same smoke test and is evaluated once on $\mathcal{D}_{\mathrm{dev}}$.

Joint refinement. Composition combines four mechanisms in one program, but it does not by itself make their interactions good. For example, a template admission rule developed against a windowed response behaves differently once the decoder changes the window, and a recovery trigger written against the seed memory can fire on observations that the new memory would have rejected. Joint refinement therefore returns to the coupled controller space: one campaign of K_2 rounds starts from π_{c}, and each revision may change any of the four operations. What distinguishes this campaign is its evidence. Besides its own archive, the agent receives the scoped archives, frozen at the end of scoped exploration: each of the $4K_1$ scoped revisions with its source, its patch, its aggregate and per-sequence development-split metrics, and the agent's own account of the mechanism. These archives are reference evidence rather than a parent set; parents remain π_{c} and the rounds of this campaign. With them, the agent can see which composed mechanism carried the gain of each space, which alternatives from the same space came close, and where a mechanism should be revised, replaced, or removed now that the others are present. In our runs, the agent used this evidence to revise interactions. For instance, it separated the windowed response used to localize the target from the raw response used as confidence, so that the template memory and the recovery trigger, which consume the confidence, are no longer affected by the spatial prior of the decoder (Appendix E).

3.4 Selection and Deployment

The final controller $\pi^\star$ is the joint-refinement round with the highest development-split AUC, selected using no held-out information. It is then evaluated unchanged, with the same program and the same hyperparameters, on every held-out benchmark. The whole procedure uses $4K_1 + K_2$

agent rounds and $4K_1 + K_2 + 1$ evaluations of $\mathcal{D}_{\text{dev}}$, one per round and one for the composite; with $K_1 = 15$ and $K_2 = 30$, this is 90 rounds and 91 development-split evaluations. At deployment, $\pi^\star$ is ordinary tracker code around the frozen visual component or model set used by the corresponding seed, with no language model, no network service, no learned network beyond that seed's frozen visual component or model set, and the same runtime inputs as that seed.

3.5 Enlarging the Frozen Model Set

Nothing in CRAFT assumes a single visual model. To test the framework on a stronger starting point, we enlarge the frozen model set to $\{f_{\boldsymbol{\theta}}, \text{SAM3}\}$, where SAM3 (Carion et al., 2026) propagates the first-frame box through the video and provides, at every frame, a box, an objectness score, a low-resolution mask, and three candidate boxes with predicted IoU. The propagation is open-loop: it depends only on the video and the initialization box, so it is a fixed observation stream that the controller reads causally under constraint (ii), and the controller cannot write back to it.

The seed controller of this tracker anchors the search crop at the SAM3 box with a smaller crop factor whenever the objectness exceeds a threshold, and otherwise falls back to the base seed. The design spaces, the constraints, and the three stages are unchanged; each space may use the stream within its own responsibility, for instance the decoder to arbitrate between the model's response and the SAM3 candidates.

4 Experiments

In this section, we evaluate how much CRAFT improves tracking through controller design, and where the gains come from. We first describe the experimental setup, then compare the resulting trackers with prior work on five benchmarks, and finally analyze the design procedure.

4.1 Experimental Setup

We apply CRAFT to Seed-B256, a one-stream Transformer tracker (Ye et al., 2022), and to Seed-SAM3, which adds SAM3 (Carion et al., 2026) to its frozen model set (Section 3.5); the development split is the TNL2K training split (Wang et al., 2021); LaSOT (Fan et al., 2019), LaSOT$_{\text{ext}}$ (Fan et al., 2021), and the TNL2K test split are the main held-out benchmarks, and the short-term benchmarks TrackingNet (Müller et al., 2018) and GOT-10k (Huang et al., 2021) are also used for evaluation. More details are provided in Appendix B.

4.2 Main Results

Held-out gains on a frozen model. Table 1 compares the seed trackers, the CRAFT trackers, and prior work. CRAFT-B256 improves over its seed on every benchmark, by 2.0 to 3.0 AUC points, with the visual model and its inference procedure unchanged and with one program, rather than a per-benchmark configuration, on all three. Precision moves with AUC, by larger margins, so the controller localizes the target more precisely as well as more often. The improvement on LaSOT and LaSOT$_{\text{ext}}$ is moreover cross-dataset, since the design procedure observed only TNL2K training videos; LaSOT$_{\text{ext}}$ further tests categories absent from LaSOT. These gains come from the controller alone and place CRAFT-B256 first among base-scale trackers on LaSOT and TNL2K, above trackers that modify the training objective (Chen et al., 2026), the pretraining (Liang et al., 2025), the encoder (Lin et al., 2024; Chen et al., 2025), or the temporal inference (Zheng et al., 2024), and level with the best LaSOT$_{\text{ext}}$ AUC. Those methods improve the model, whereas CRAFT improves the program around it, so the two kinds of gain are complementary in principle.

A stronger starting point. Seed-SAM3 starts well above Seed-B256 because SAM3 propagation supplies a reliable search anchor. CRAFT improves this tracker as well, by 1.2 to 2.2 AUC points, to the highest AUC of its group on all three benchmarks and clearly above standalone SAM3. The SAM-based trackers of this group (Videnovic et al., 2025; Ding et al., 2025; Yang et al., 2026) also keep their model frozen and gain from hand-designed motion and memory logic; CRAFT obtains logic of this kind, for a different model set, from the design procedure. The procedure is thus not tied to the weaknesses of one seed: the headroom on Seed-SAM3 lies in different design spaces than

Table 1: **Comparison on three large-scale tracking benchmarks.** Base-scale trackers are in the upper block and trackers built on foundation models in the lower block. Seed-B256 and Seed-SAM3 are the seed trackers; CRAFT-B256 and CRAFT-SAM3 are the same frozen models with the controller returned by CRAFT. AUC, P, and P_{Norm} denote area under the success curve, precision, and normalized precision. Within each block, **bold** and underlined values indicate the best and second-best results, respectively; a dash marks a result not reported under RGB-only evaluation.

Tracker	LaSOT			$\text{LaSOT}_{\text{ext}}$			TNL2K		
	AUC	P	P_{Norm}	AUC	P	P_{Norm}	AUC	P	P_{Norm}
OSTrack-B256 (Ye et al., 2022)	69.1	75.2	78.7	47.4	53.3	57.3	54.3	56.3	72.5
ODTrack-B384 (Zheng et al., 2024)	73.2	80.6	83.2	52.4	60.1	63.9	60.9	64.5	79.1
LoRAT-B224 (Lin et al., 2024)	71.7	77.3	80.9	50.3	57.1	61.6	58.8	61.3	76.3
SUTrack-B224 (Chen et al., 2025)	73.2	80.5	83.4	53.1	60.5	64.2	–	–	–
ARPTrack-B256 (Liang et al., 2025)	72.6	78.5	81.4	52.0	58.7	62.9	–	–	–
RELO-B256 (Chen et al., 2026)	73.3	80.6	83.4	**54.2**	**62.2**	65.4	60.9	64.3	77.5
Seed-B256 (seed)	72.6	80.4	82.4	52.0	59.0	63.0	60.6	63.6	77.4
CRAFT-B256 (Ours)	**74.6**	**82.9**	**84.8**	**54.2**	61.7	**65.7**	**63.6**	**68.0**	**81.1**
DAM4SAM-L (Videnovic et al., 2025)	75.1	–	–	60.9	–	–	–	–	–
SAM2Long (Ding et al., 2025)	73.9	–	–	60.9	–	–	–	–	–
SAMURAI-L (Yang et al., 2026)	74.2	80.2	82.7	61.0	72.2	73.9	61.0	–	–
SAM3 (Carion et al., 2026)	75.5	79.7	82.7	64.0	75.3	76.8	65.9	72.9	83.0
Seed-SAM3 (seed)	77.6	**86.7**	88.1	63.8	75.9	76.2	66.1	71.8	83.7
CRAFT-SAM3 (Ours)	**78.8**	85.9	**90.2**	**65.9**	**77.5**	**78.9**	**68.3**	**75.9**	**86.1**

Table 2: **Ablation results of CRAFT.** Results report AUC on the development split (Dev.) and on the three held-out benchmarks. Rows 2 and 3 replace scoped exploration by an unscoped or a sequential campaign at the same budget; rows 4 and 5 remove joint refinement or its scoped evidence. Δ denotes the average AUC change over the four splits relative to the full procedure.

#	Variant	Dev.	LaSOT	$\text{LaSOT}_{\text{ext}}$	TNL2K	Δ
1	Full CRAFT (CRAFT-B256)	68.8	74.6	54.2	63.6	–
2	Single unscoped campaign	67.4	73.5	52.9	62.3	-1.3
3	Sequential scoped campaigns	68.6	74.4	54.0	62.9	-0.3
4	w/o joint refinement	68.2	74.2	52.1	63.5	-0.8
5	w/o scoped archives in refinement	68.3	74.3	53.8	63.4	-0.4

on Seed-B256 (Appendix D), and the same three stages exploit it. The CRAFT controller leaves the speed of both trackers nearly unchanged (Table 4); for Seed-SAM3 it is set by SAM3 itself.

Short-term benchmarks. The same programs also improve both trackers on TrackingNet and GOT-10k; CRAFT-SAM3 leads its group on every metric. Details are in Appendix C.

4.3 Ablation and Analysis

We first trace the gains from the scoped revisions to the deployed program, then ablate the design procedure on Seed-B256 stage by stage (Table 2), and end with the discovered controllers.

From scoped revisions to the deployed program. Following the AUC through the three stages shows that scoped exploration locates headroom in different design spaces for the two trackers, that composition merges the gains and refinement resolves their interactions, and that development-split AUC predicts held-out AUC, with Pearson coefficients of 0.82 to 0.97 across the programs of the Seed-B256 run. Stage-by-stage results and round-by-round trajectories are in Appendix D.

Decomposition into design spaces. Row 2 of Table 2 replaces the four scoped campaigns by a single unscoped campaign from the seed at the same budget, in which every revision may change any design space. It is the weakest variant, more than a point below CRAFT on every split. This is the situation anticipated in Section 3.1: when a revision may change several spaces, its evaluation cannot be attributed to any of them, and the campaign gains less usable evidence per evaluation.

Independent versus sequential campaigns. Row 3 keeps the scoping but runs the four campaigns in sequence rather than in parallel, in the order in which the operations occur within a frame, each for 15 rounds from the best program of the previous space, followed by the same 30-round joint refinement with the chained archives as its evidence. This chain closes most of the gap but remains below CRAFT on the four splits, and the difference lies in the evidence that reaches joint refinement. In the chain, the four archives were measured against four different starting programs, so the per-sequence results of one space cannot be compared with those of another, and the refinement agent has a weaker basis for deciding which mechanism to revise; in CRAFT, every archive is measured against the common seed, so the scoped archives tell the agent what each space contributed on which sequences. The small size of the gap shows that most of the benefit comes from the decomposition itself and survives a change of schedule; the remainder is what comparable scoped evidence recovers.

Joint refinement. Row 4 removes joint refinement and spends its rounds on scoped exploration instead, 23 per design space rather than 15, so the budget is nearly unchanged; the deployed composite trails the final program on every split and, most of all, on $\text{LaSOT}_{\text{ext}}$. Longer scoped campaigns do not substitute for refinement, which is what makes the composed mechanisms work together.

Evidence from scoped exploration. Row 5 keeps the thirty refinement rounds but withholds the scoped archives, so the agent sees the composite and its own history only. Without this record, it falls short of the full procedure on the development split and on every held-out benchmark, slightly more than the chained archives of row 3 cost. The scoped campaigns thus provide two benefits: the mechanisms that enter the composite, and the evidence that lets refinement revise them well.

What the agent discovered. The returned program is readable tracker code, and two unprompted patterns recur across the accepted rounds: verification before a state change, and calibration per object rather than per dataset. More details of the discovered controllers are provided in Appendix E.

5 Conclusion and Discussion

We have presented CRAFT, which makes the tracking controller an optimizable component of the tracker. Coding agents propose and test revisions of the controller offline from development-split feedback, under constraints that keep the result deployable, while the visual model and its inference procedure stay fixed. CRAFT organizes this design around four purpose-constrained design tasks. Scoped exploration locates the headroom of each task with evidence attributable to that task, metric-blind composition integrates the four selected mechanisms into one program, and joint refinement re-tunes them to one another. The output is ordinary tracker code that runs at nearly the speed of the seed, with no language model at inference. With the TNL2K training split as its only source of feedback, CRAFT improves every held-out benchmark for a base-scale one-stream tracker and its SAM3-augmented counterpart, with one program per tracker throughout. Since the visual models are untouched, these gains come from the controller alone and complement advances in the model.

Beyond the benchmark gains, the runs support three observations. First, the gains transfer: the improvement holds on every held-out benchmark, and development-split AUC tracked held-out AUC across the programs of a run, so selection on the development split alone sufficed. Second, where a tracker can still improve is not known in advance: the headroom lay in template-memory management for the one-stream tracker and spread toward response decoding once SAM3 anchored the search crop, and scoped exploration is what located it. Third, the discovered programs are readable tracker code in which two unprompted patterns recur, verification before a state change and calibration per object rather than per dataset, and the latter is consistent with the cross-dataset transfer.

These results also clarify the scope of this study: the four operation-level design tasks are fixed before a run and the visual model is frozen, and both are natural places to extend the framework. The tasks could be proposed by the agent from failure analysis. The controller and the model could also be improved in alternation, with the discovered controller defining the deployment conditions for training the next model. More broadly, CRAFT assumes a frozen perception model, a controller written as code, and an automatic evaluator on a development split. These assumptions hold in other tracking settings where online decision logic surrounds a frozen model, such as segmentation trackers, multi-object and multi-camera tracking, and vision-language tracking. Applying CRAFT to these settings is a natural next step.

AI USE STATEMENT

We used generative AI tools to assist with code implementation, in two ways. First, the coding agent is the object of study: the tracking controllers reported in Section 4 were written by it inside the design procedure of Section 3. This procedure was conceived, specified, and controlled by the authors, who formulated the problem and the framework, defined the four design spaces, the constraints on admissible programs, the automatic evaluator, the round budgets, and the selection rule, and chose the base trackers, the development split, and the benchmarks; the agent proposed and implemented controller revisions only within this procedure, and no agent had access to any held-out benchmark. Second, we used AI coding assistants to speed up the implementation of the CRAFT framework itself, the driver of the design procedure and its evaluation infrastructure, from designs and specifications written by the authors. Beyond these two uses, we have not used generative AI tools to develop theoretical models or conceptual frameworks, to propose or refine hypotheses, to design or provide feedback on research methodology or experiments, to clean or reformat datasets, to support qualitative or thematic data analysis, to interpret results, or to assist with translation; generating synthetic datasets and formulating or proving mathematical claims are not applicable to this work. Additionally, we used AI tools to edit the text written by the authors for readability.

We have reviewed all AI-assisted work. Every agent-written revision was archived with its full diff and transcript and executed only through the automatic evaluator; the final controllers were re-read line by line by the authors, and the mechanisms summarized in Appendix E are the authors' account of that reading. The framework code written with AI assistance was carefully reviewed and tested by the authors before use. All results were obtained and interpreted by the authors. Every AI-assisted edit to the text was checked by the authors. We take responsibility for the final content of this work, including text, claims, or artifacts produced with the aid of generative AI.

ETHICS STATEMENT

This work improves visual object tracking, which has beneficial uses in robotics, sports analysis, and human-computer interaction, and, like all tracking methods, could be misused for surveillance. The procedure automates part of the algorithm-design process; we archive and review all agent-written code and encourage the same practice in derived work.

REPRODUCIBILITY STATEMENT

The design procedure, design-space definitions, agent constraints, and selection rule are specified in Section 3; the base trackers, data splits, round budgets, and agent settings are specified in Section 4.1 and Appendix B. The development split is a fixed, published list of TNL2K training sequences, and the seed controller is the default tracker code of the base model. We will release the seed tracker, the driver of the design procedure, the frozen SAM3 observation streams, every archived round (source snapshot, patch, metrics, and transcript), and the final tracking controllers, so that both the deployed trackers and the design procedure itself can be reproduced.

REFERENCES

Yifan Bai, Zeyang Zhao, Yihong Gong, and Xing Wei. ARTrackV2: Prompting autoregressive tracker where to look and how to describe. In *CVPR*, pp. 19048–19057, 2024.

Luca Bertinetto, Jack Valmadre, João F. Henriques, Andrea Vedaldi, and Philip H. S. Torr. Fully-convolutional Siamese networks for object tracking. In *ECCV Workshops*, pp. 850–865, 2016.

Goutam Bhat, Martin Danelljan, Luc Van Gool, and Radu Timofte. Learning discriminative model prediction for tracking. In *ICCV*, pp. 6182–6191, 2019.

Nicolas Carion, Laura Gustafson, Yuan-Ting Hu, Shoubhik Debnath, Ronghang Hu, Didac Suris, Chaitanya Ryali, Kalyan Vasudev Alwala, Haitham Khedr, Andrew Huang, et al. SAM 3: Segment anything with concepts. In *ICLR*, pp. 138846–138923, 2026.

Jun Shern Chan, Neil Chowdhury, Oliver Jaffe, James Aung, Dane Sherburn, Evan Mays, Giulio Starace, Kevin Liu, Leon Maksin, Tejal Patwardhan, et al. MLE-bench: Evaluating machine learning agents on machine learning engineering. In *ICLR*, pp. 50466–50494, 2025.

Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, et al. Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374*, 2021a.

Xin Chen, Bin Yan, Jiawen Zhu, Dong Wang, Xiaoyun Yang, and Huchuan Lu. Transformer tracking. In *CVPR*, pp. 8126–8135, 2021b.

Xin Chen, Houwen Peng, Dong Wang, Huchuan Lu, and Han Hu. SeqTrack: Sequence to sequence learning for visual object tracking. In *CVPR*, pp. 14572–14581, 2023.

Xin Chen, Ben Kang, Wanting Geng, Jiawen Zhu, Yi Liu, Dong Wang, and Huchuan Lu. SUTrack: Towards simple and unified single object tracking. In *AAAI*, pp. 2239–2247, 2025.

Xin Chen, Chuanyu Sun, Jiao Xu, Houwen Peng, Dong Wang, Huchuan Lu, and Kede Ma. RELO: Reinforcement learning to localize for visual object tracking. In *ICML*, 2026.

Yutao Cui, Cheng Jiang, Limin Wang, and Gangshan Wu. MixFormer: End-to-end tracking with iterative mixed attention. In *CVPR*, pp. 13608–13618, 2022.

Kenan Dai, Yunhua Zhang, Dong Wang, Jianhua Li, Huchuan Lu, and Xiaoyun Yang. High-performance long-term tracking with meta-updater. In *CVPR*, pp. 6297–6306, 2020.

Martin Danelljan, Goutam Bhat, Fahad Shahbaz Khan, and Michael Felsberg. ATOM: Accurate tracking by overlap maximization. In *CVPR*, pp. 4660–4669, 2019.

Shuangrui Ding, Rui Qian, Xiaoyi Dong, Pan Zhang, Yuhang Zang, Yuhang Cao, Yuwei Guo, Dahua Lin, and Jiaqi Wang. SAM2Long: Enhancing SAM 2 for long video segmentation with a training-free memory tree. In *ICCV*, pp. 13614–13624, 2025.

Xingping Dong, Jianbing Shen, Wenguan Wang, Ling Shao, Haibin Ling, and Fatih Porikli. Dynamical hyperparameter optimization via deep reinforcement learning in tracking. *IEEE TPAMI*, 43(5):1515–1529, 2021.

Heng Fan, Liting Lin, Fan Yang, Peng Chu, Ge Deng, Sijia Yu, Hexin Bai, Yong Xu, Chunyuan Liao, and Haibin Ling. LaSOT: A high-quality benchmark for large-scale single object tracking. In *CVPR*, pp. 5374–5383, 2019.

Heng Fan, Hexin Bai, Liting Lin, Fan Yang, Peng Chu, Ge Deng, Sijia Yu, Harshit, Mingzhen Huang, Juehuan Liu, et al. LaSOT: A high-quality large-scale single object tracking benchmark. *IJCV*, 129(2):439–461, 2021.

Xiaokun Feng, Shiyu Hu, Xuchen Li, Dailing Zhang, Meiqi Wu, Jing Zhang, Xiaotang Chen, and Kaiqi Huang. ATCTrack: Aligning target-context cues with dynamic target states for robust vision-language tracking. In *ICCV*, pp. 19850–19861, 2025.

Lingyi Hong, Shilin Yan, Renrui Zhang, Wanyun Li, Xinyu Zhou, Pinxue Guo, Kaixun Jiang, Yiting Chen, Jinglun Li, Zhaoyu Chen, et al. OneTracker: Unifying visual object tracking with foundation models and efficient tuning. In *CVPR*, pp. 19079–19091, 2024.

Zhibin Hong, Zhe Chen, Chaohui Wang, Xue Mei, Danil Prokhorov, and Dacheng Tao. MUlti-Store Tracker (MUSTer): A cognitive psychology inspired approach to object tracking. In *CVPR*, pp. 749–758, 2015.

Lianghua Huang, Xin Zhao, and Kaiqi Huang. GlobalTrack: A simple and strong baseline for long-term tracking. In *AAAI*, pp. 11037–11044, 2020.

Lianghua Huang, Xin Zhao, and Kaiqi Huang. GOT-10k: A large high-diversity benchmark for generic object tracking in the wild. *IEEE TPAMI*, 43(5):1562–1577, 2021.

Yuqing Huang, Guotian Zeng, Zhenqiao Yuan, Zhenyu He, Xin Li, Yaowei Wang, and Ming-Hsuan Yang. Interactive tracking: A human-in-the-loop paradigm with memory-augmented adaptation. In *CVPR*, pp. 35166–35176, 2026.

Zhengyao Jiang, Dominik Schmidt, Dhruv Srikanth, Dixing Xu, Ian Kaplan, Deniss Jacenko, and Yuxiang Wu. AIDE: AI-driven exploration in the space of code. *arXiv preprint arXiv:2502.13138*, 2025.

Carlos E. Jimenez, John Yang, Alexander Wettig, Shunyu Yao, Kexin Pei, Ofir Press, and Karthik Narasimhan. SWE-bench: Can language models resolve real-world GitHub issues? In *ICLR*, pp. 54107–54157, 2024.

Zdenek Kalal, Krystian Mikolajczyk, and Jiri Matas. Tracking-learning-detection. *IEEE TPAMI*, 34 (7):1409–1422, 2012.

Joel Lehman, Jonathan Gordon, Shawn Jain, Kamal Ndousse, Cathy Yeh, and Kenneth O. Stanley. Evolution through large models. In Wolfgang Banzhaf, Penousal Machado, and Mengjie Zhang (eds.), *Handbook of Evolutionary Machine Learning*, pp. 331–366. Springer, 2024.

Bo Li, Junjie Yan, Wei Wu, Zheng Zhu, and Xiaolin Hu. High performance visual tracking with Siamese region proposal network. In *CVPR*, pp. 8971–8980, 2018.

Xiaohai Li, Bineng Zhong, Qihua Liang, Zhiyi Mo, Jian Nong, and Shuxiang Song. Dynamic updates for language adaptation in visual-language tracking. In *CVPR*, pp. 19165–19174, 2025.

Shiyi Liang, Yifan Bai, Yihong Gong, and Xing Wei. Autoregressive sequential pretraining for visual tracking. In *CVPR*, pp. 7254–7264, 2025.

Liting Lin, Heng Fan, Zhipeng Zhang, Yong Xu, and Haibin Ling. SwinTrack: A simple and strong baseline for Transformer tracking. In *NeurIPS*, pp. 16743–16754, 2022.

Liting Lin, Heng Fan, Zhipeng Zhang, Yaowei Wang, Yong Xu, and Haibin Ling. Tracking meets LoRA: Faster training, larger model, stronger performance. In *ECCV*, pp. 300–318, 2024.

Tsung-Yi Lin, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C. Lawrence Zitnick. Microsoft COCO: Common objects in context. In *ECCV*, pp. 740–755, 2014.

Fei Liu, Xialiang Tong, Mingxuan Yuan, Xi Lin, Fu Luo, Zhenkun Wang, Zhichao Lu, and Qingfu Zhang. Evolution of heuristics: Towards efficient automatic algorithm design using large language model. In *ICML*, pp. 32201–32223, 2024.

Hanxiao Liu, Karen Simonyan, and Yiming Yang. DARTS: Differentiable architecture search. In *ICLR*, 2019.

Chris Lu, Cong Lu, Robert Tjarko Lange, Yutaro Yamada, Shengran Hu, Jakob Foerster, David Ha, and Jeff Clune. Towards end-to-end automation of AI research. *Nature*, 651(8107):914–919, 2026.

Bruce D. Lucas and Takeo Kanade. An iterative image registration technique with an application to stereo vision. In *IJCAI*, pp. 674–679, 1981.

Chao Ma, Xiaokang Yang, Chongyang Zhang, and Ming-Hsuan Yang. Long-term correlation tracking. In *CVPR*, pp. 5388–5396, 2015.

Yecheng Jason Ma, William Liang, Guanzhi Wang, De-An Huang, Osbert Bastani, Dinesh Jayaraman, Yuke Zhu, Linxi Fan, and Anima Anandkumar. Eureka: Human-level reward design via coding large language models. In *ICLR*, pp. 26516–26560, 2024.

Christoph Mayer, Martin Danelljan, Danda Pani Paudel, and Luc Van Gool. Learning target candidate association to keep track of what not to track. In *ICCV*, pp. 13444–13454, 2021.

Christoph Mayer, Martin Danelljan, Goutam Bhat, Matthieu Paul, Danda Pani Paudel, Fisher Yu, and Luc Van Gool. Transforming model prediction for tracking. In *CVPR*, pp. 8731–8740, 2022.

Matthias Müller, Adel Bibi, Silvio Giancola, Salman Alsubaihi, and Bernard Ghanem. TrackingNet: A large-scale dataset and benchmark for object tracking in the wild. In *ECCV*, pp. 300–317, 2018.

Alexander Novikov, Ngân Vũ, Marvin Eisenberger, Emilien Dupont, Po-Sen Huang, Adam Zsolt Wagner, Sergey Shirobokov, Borislav Kozlovskii, Francisco J. R. Ruiz, Abbas Mehrabian, et al. AlphaEvolve: A coding agent for scientific and algorithmic discovery. *arXiv preprint arXiv:2506.13131*, 2025.

Liang Peng, Junyuan Gao, Xinran Liu, Weihong Li, Shaohua Dong, Zhipeng Zhang, Heng Fan, and Libo Zhang. VastTrack: Vast category visual object tracking. In *NeurIPS*, pp. 130797–130818, 2024.

Nikhila Ravi, Valentin Gabeur, Yuan-Ting Hu, Ronghang Hu, Chaitanya Ryali, Tengyu Ma, Haitham Khedr, Roman Rädle, Chloe Rolland, Laura Gustafson, et al. SAM 2: Segment anything in images and videos. In *ICLR*, pp. 28085–28128, 2025.

Esteban Real, Chen Liang, David So, and Quoc Le. AutoML-Zero: Evolving machine learning algorithms from scratch. In *ICML*, pp. 8007–8019, 2020.

Bernardino Romera-Paredes, Mohammadamin Barekatain, Alexander Novikov, Matej Balog, M. Pawan Kumar, Emilien Dupont, Francisco J. R. Ruiz, Jordan S. Ellenberg, Pengming Wang, Omar Fawzi, et al. Mathematical discoveries from program search with large language models. *Nature*, 625(7995):468–475, 2024.

Tong Shen, Di Wang, and José M. F. Moura. MSRTrack: LLM-powered object tracking with motion and semantic reasoning. In *WACV*, pp. 838–848, 2026.

Ke Song, Wei Zhang, Ran Song, and Yibin Li. Online decision based visual tracking via reinforcement learning. In *NeurIPS*, pp. 11778–11788, 2020.

Yiming Sun, Fan Yu, Shaoxiang Chen, Yu Zhang, Junwei Huang, Yang Li, Chenhui Li, and Changbo Wang. ChatTracker: Enhancing visual tracking performance via chatting with multimodal large language model. In *NeurIPS*, pp. 39303–39324, 2024.

Yunjie Tian, Lingxi Xie, Jihao Qiu, Jianbin Jiao, Yaowei Wang, Qi Tian, and Qixiang Ye. Fast-iTPN: Integrally pre-trained Transformer pyramid network with token migration. *IEEE TPAMI*, 46(12):9766–9779, 2024.

Jovana Videnovic, Alan Lukezic, and Matej Kristan. A distractor-aware memory for visual object tracking with SAM2. In *CVPR*, pp. 24255–24264, 2025.

Xiao Wang, Xiujun Shu, Zhipeng Zhang, Bo Jiang, Yaowei Wang, Yonghong Tian, and Feng Wu. Towards more flexible and accurate object tracking with natural language: Algorithms and benchmark. In *CVPR*, pp. 13763–13773, 2021.

Xing Wei, Yifan Bai, Yongchao Zheng, Dahu Shi, and Yihong Gong. Autoregressive visual tracking. In *CVPR*, pp. 9697–9706, 2023.

Yi Wu, Jongwoo Lim, and Ming-Hsuan Yang. Online object tracking: A benchmark. In *CVPR*, pp. 2411–2418, 2013.

Bin Yan, Haojie Zhao, Dong Wang, Huchuan Lu, and Xiaoyun Yang. 'Skimming-Perusal' Tracking: A framework for real-time and robust long-term tracking. In *ICCV*, pp. 2385–2393, 2019.

Bin Yan, Houwen Peng, Jianlong Fu, Dong Wang, and Huchuan Lu. Learning spatio-temporal Transformer for visual tracking. In *ICCV*, pp. 10448–10457, 2021.

Cheng-Yen Yang, Hsiang-Wei Huang, Wenhao Chai, Zhongyu Jiang, and Jenq-Neng Hwang. SAMURAI: Motion-aware memory for training-free visual object tracking with SAM 2. *IEEE TIP*, 35:970–982, 2026.

Chengrun Yang, Xuezhi Wang, Yifeng Lu, Hanxiao Liu, Quoc V. Le, Denny Zhou, and Xinyun Chen. Large language models as optimizers. In *ICLR*, pp. 12028–12068, 2024.

Botao Ye, Hong Chang, Bingpeng Ma, Shiguang Shan, and Xilin Chen. Joint feature learning and relation modeling for tracking: A one-stream framework. In *ECCV*, pp. 341–357, 2022.

Lichao Zhang, Abel Gonzalez-Garcia, Joost van de Weijer, Martin Danelljan, and Fahad Shahbaz Khan. Learning the model update for Siamese trackers. In *ICCV*, pp. 4010–4019, 2019.

Fei Zhao, Ting Zhang, Yibing Song, Ming Tang, Xiaobo Wang, and Jinqiao Wang. Siamese regression tracking with reinforced template updating. *IEEE TIP*, 30:628–640, 2021.

Yaozong Zheng, Bineng Zhong, Qihua Liang, Zhiyi Mo, Shengping Zhang, and Xianxian Li. ODTrack: Online dense temporal token learning for visual tracking. In *AAAI*, pp. 7588–7596, 2024.

Barret Zoph and Quoc V. Le. Neural architecture search with reinforcement learning. In *ICLR*, 2017.

A Design-Space Definitions

Table 3 lists the four design spaces of the tracking controller as they were defined for the agents. Every round of scoped exploration received these definitions together with two instructions: spaces are delimited by the primary causal purpose of a mechanism, not by a file or function partition, and the parent's behavior in the other spaces must be preserved. The last column gives the realization of each space in the seed controller of Seed-B256; the seed controller of Seed-SAM3 differs only in search-region selection, as described in Section 3.5.

Table 3: **Design spaces of the tracking controller.** The scope column is the definition given to the agents in every round of scoped exploration, with the names used in this paper.

Design space	Scope	Seed controller
Search-region selection	Normal-tracking observation and search-region generation: where to look, crop center and extent, scale or multi-region coverage, and causal motion-guided local search.	Fixed factor around the previous box.
Response decoding	Normal-tracking model-output interpretation and state update: response calibration, window or scale penalties, candidate selection and aggregation, box smoothing, and acceptance of the current measurement.	Cosine window, box at the maximum.
Template-memory management	Appearance memory and template management: collection and quality gating, storage and eviction, selection and fusion, and protection against template contamination.	Confidence- and interval-gated update.
Failure recovery	Loss suspicion and recovery as one coupled mechanism: entry and exit conditions, enlarged or global and coarse-to-fine search, candidate validation, and causal re-acquisition.	None.

B Experimental Setup Details

Section 4.1 names the two base trackers, the development split, and the held-out benchmarks. Here, we provide the details of the trackers and their deployed cost, the data splits and metrics, and the agent and budget of a CRAFT run.

Table 4: **Trackers and deployed cost.** Parameters and FLOPs refer to the frozen models, which CRAFT does not change; for Seed-SAM3, parentheses give the SAM3 component alone. Speed is measured on a single NVIDIA A40 GPU with the seed controller and with the CRAFT controller.

Tracker	# Params (M)	FLOPs (G)	Speed (FPS)	
			Seed	CRAFT
Seed-B256 / CRAFT-B256	70	39	50	47
Seed-SAM3 / CRAFT-SAM3	930 (860)	3439 (3400)	7 (8)	7 (8)

Base trackers. Seed-B256 is a one-stream Transformer tracker in the OSTrack framework (Ye et al., 2022) with a Fast-iTPN-B encoder (Tian et al., 2024), a 256×256 search crop, 128×128 templates, a patch stride of 16, four template slots, and a center-based prediction head. It is trained on the training splits of COCO (Lin et al., 2014), LaSOT (Fan et al., 2019), GOT-10k (Huang et al., 2021), TrackingNet (Müller et al., 2018), and VastTrack (Peng et al., 2024), and its weights are frozen throughout. Its seed controller is the one described in Section 3.1: a search factor of 4, a cosine window, the box at the maximum, and a template update every 50 frames when the confidence exceeds 0.90. Seed-SAM3 enlarges the frozen model set with SAM3 (Carion et al., 2026) as described in Section 3.5, using the default SAM3 video configuration, an objectness threshold of 0.5, and a crop factor of 3 around the SAM3 box; the SAM3 stream is precomputed once per sequence and read causally. Table 4 lists the deployed cost of both trackers with the seed controller and with the controller returned by CRAFT.

Development split and held-out benchmarks. The development split $\mathcal{D}_{\text{dev}}$ consists of the 869 RGB sequences of the TNL2K training split (Wang et al., 2021), 0.51M frames in total. It is disjoint from the training data of the base tracker and from every benchmark we report. The held-out benchmarks are the LaSOT test split (Fan et al., 2019), LaSOT$_{\text{ext}}$ (Fan et al., 2021), a test-only set whose categories are disjoint from LaSOT, and the TNL2K test split; LaSOT and LaSOT$_{\text{ext}}$ are therefore also cross-dataset with respect to the development split. We additionally evaluate the selected programs on the short-term benchmarks TrackingNet (Müller et al., 2018) and GOT-10k (Huang et al., 2021) through their evaluation servers. Every benchmark is evaluated with the same program and the same hyperparameters; no hyperparameter is adjusted per benchmark. We report the area under the success curve (AUC) as the primary metric, together with precision (P) and normalized precision (P_{Norm}), and on GOT-10k the average overlap (AO) and the success rates at IoU thresholds 0.5 and 0.75.

Agent and budget. Each round is executed by a coding agent in a fresh session under the constraints of Section 3.1; every round, for both trackers, uses GPT-5.6-sol at high reasoning effort. Scoped exploration runs $K_1 = 15$ rounds per design space and joint refinement runs $K_2 = 30$ rounds, so one CRAFT run comprises 90 rounds and, including the composite, 91 development-split evaluations. Each campaign runs on four GPUs: a development-split evaluation takes about 25 minutes with 16 workers and an agent proposal about 4 minutes, so one round costs about half an hour. All held-out evaluations were run after the design procedure had finished. No held-out result entered any agent's input or any selection.

C Results on Short-Term Benchmarks

Table 5: **Comparison on two short-term tracking benchmarks.** AUC, P, P_{Norm}, AO, and SR denote area under the success curve, precision, normalized precision, average overlap, and success rate. Blocks, seed rows, and markings follow Table 1.

Tracker	TrackingNet			GOT-10k		
	AUC	P	P_{Norm}	AO	$\text{SR}_{0.5}$	$\text{SR}_{0.75}$
OSTrack-B256 (Ye et al., 2022)	83.1	82.0	87.8	71.0	80.4	68.2
SeqTrack-B256 (Chen et al., 2023)	83.3	82.2	88.3	74.7	84.7	71.8
ARTrack-B256 (Wei et al., 2023)	84.2	83.5	88.7	73.5	82.2	70.9
ARTrackV2-B256 (Bai et al., 2024)	84.9	84.5	89.3	75.9	85.4	72.7
ODTrack-B384 (Zheng et al., 2024)	85.1	84.9	90.1	77.0	87.9	75.1
LoRAT-B224 (Lin et al., 2024)	83.5	82.1	87.9	72.1	81.8	70.7
SUTrack-B224 (Chen et al., 2025)	85.7	85.1	90.3	77.9	87.5	78.5
RELO-B256 (Chen et al., 2026)	**86.4**	**86.2**	90.7	**80.5**	**90.5**	**81.2**
Seed-B256 (seed)	84.7	84.3	89.3	76.5	85.9	76.4
CRAFT-B256 (Ours)	86.3	**86.2**	**90.8**	79.5	89.2	80.1
DAM4SAM-L (Videnovic et al., 2025)	–	–	–	81.1	–	–
SAM2Long (Ding et al., 2025)	–	–	–	81.1	–	–
SAMURAI-L (Yang et al., 2026)	85.3	–	–	81.7	92.2	76.9
SAM3 (Carion et al., 2026)	86.6	88.4	91.4	84.0	95.0	82.5
Seed-SAM3 (seed)	85.9	86.4	90.8	84.4	95.0	83.2
CRAFT-SAM3 (Ours)	**87.3**	**89.3**	**92.2**	**85.9**	**95.9**	**86.1**

Section 4.2 reports the held-out results on three large-scale long-term benchmarks. Here, we provide the results of the same programs on two short-term benchmarks. Table 5 reports TrackingNet and GOT-10k, on which the same unchanged programs are evaluated through the official servers. Both trackers improve over their seeds on both benchmarks, and on GOT-10k the strict-overlap success rate $\text{SR}_{0.75}$ rises the most, so the discovered controllers produce tighter boxes rather than only fewer failures. CRAFT-SAM3 obtains the best result of its group on every metric. The controller gains thus carry over from the long-term benchmarks to large-scale short-term ones, with the same program and the same hyperparameters.

Table 6: **From scoped revisions to the deployed program.** AUC (%) on the development split (Dev.) and on the three held-out benchmarks at each stage of the procedure: the seed controller, the best revision of each design space, the composite, and the final program. Held-out columns were evaluated after the procedure had finished and played no role in any selection. Δ is the AUC change relative to the seed, averaged over the four splits; **bold** marks the largest average gain.

Program	Dev.	LaSOT	$\text{LaSOT}_{\text{ext}}$	TNL2K	Δ
Seed-B256					
Seed controller	66.1	72.6	52.0	60.6	–
Search-region selection (best of 15)	66.4	72.2	53.0	60.8	+0.3
Response decoding (best of 15)	66.1	72.2	52.7	60.6	+0.1
Template-memory management (best of 15)	68.8	74.3	54.4	62.8	+2.2
Failure recovery (best of 15)	66.2	72.5	52.2	60.6	+0.0
Composite	68.1	74.2	51.9	63.9	+1.7
Final program (best of 30)	68.8	74.6	54.2	63.6	**+2.5**
Seed-SAM3					
Seed controller	70.2	77.6	63.8	66.1	–
Search-region selection (best of 15)	70.3	77.6	63.9	66.0	+0.0
Response decoding (best of 15)	71.3	77.3	64.4	67.2	+0.6
Template-memory management (best of 15)	71.5	79.2	65.1	67.6	+1.4
Failure recovery (best of 15)	70.2	77.6	63.9	66.1	+0.0
Composite	72.4	78.8	65.9	68.2	+1.9
Final program (best of 30)	72.5	78.8	65.9	68.3	**+2.0**

D From Scoped Revisions to the Deployed Program

Section 4.3 summarizes how the gains of CRAFT arise from the scoped revisions and how well development-split AUC predicts held-out AUC. Here, we provide the complete results.

Table 6 follows the AUC from the seed through the three stages, on the development split, which alone guided the procedure, and on the held-out benchmarks, which played no role in it; Figure 3(a) traces the development-split AUC of the Seed-B256 run round by round.

Scoped exploration locates the headroom. The two trackers place their headroom in different design spaces. On Seed-B256 it is concentrated in template-memory management: fifteen rounds confined to this space account for most of the final gain on every split. What the campaign found is a memory design rather than a better threshold, with admission confirmed over two consecutive frames, a bank that keeps recent and high-quality older templates and protects the first-frame template, and a trial run of each new template set on the current search crop before it replaces the active one. The other three spaces gain less in isolation: search-region selection is the second strongest space, response decoding adds a little, and failure recovery, which acts only after a loss, leaves little trace in the aggregate. On Seed-SAM3 the headroom is less concentrated and its second source moves: SAM3 already anchors the search crop, so search-region selection is nearly exhausted, whereas response decoding, which now arbitrates between the model's response and the SAM3 candidates, becomes the second strongest space. None of this is known before the run. Scoped exploration is what reveals where a given tracker can still improve and where it cannot, the answer differs between trackers, and the spaces that gain less in isolation still enter the final program, whose average gain exceeds that of the best single space on both trackers.

Composition merges the gains, refinement resolves the interactions. On Seed-B256, the composite raises development-split and TNL2K AUC well above the seed, but its $\text{LaSOT}_{\text{ext}}$ AUC does not: the four mechanisms, each developed against the seed, interact once they share the tracker state. Joint refinement is designed for this case. Given the scoped archives, the agent restructured how confidence is measured across the composed mechanisms (Appendix E) and, over thirty rounds, raised development-split AUC further, turned the $\text{LaSOT}_{\text{ext}}$ regression into a gain over the seed, and improved LaSOT while keeping the TNL2K gain. The final program is therefore not the best single-space revision plus glue but a program whose parts were re-tuned to each other. On Seed-SAM3, where the headroom is spread over two spaces, the composite already exceeds every single-space revision on the development split, $\text{LaSOT}_{\text{ext}}$, and TNL2K, showing that the mechanisms are largely

complementary and that the composing agent preserved each of them; joint refinement then has less to resolve and confirms this program with a small further gain.

Development-split AUC predicts held-out AUC. Every selection in CRAFT, from the choice of parents to the final program, uses the development split alone, so the procedure relies on this split being a faithful proxy of the held-out benchmarks. Figure 3(b) shows that it is. The Seed-B256 run produced 91 programs, of which 73 were evaluated on the held-out benchmarks: the composite and every round that improved on its starting point, which are the only programs that selection by development-split AUC could choose. Across these programs, development-split AUC correlates with held-out AUC with Pearson coefficients of 0.82 to 0.97 (Spearman 0.76 to 0.81). These held-out evaluations were run after the procedure had finished, and none of their results reached any agent or any selection.

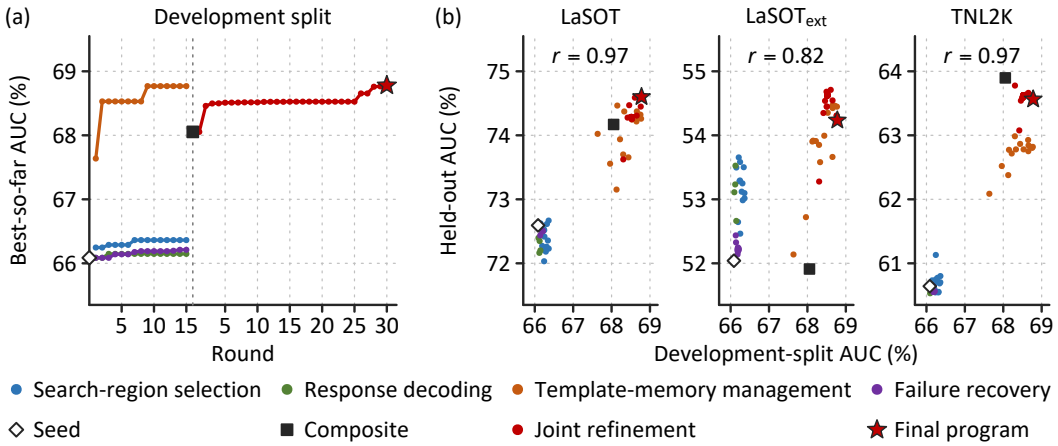


Figure 3: **Design trajectories and development-to-held-out agreement on Seed-B256.** (a) Best-so-far development-split AUC over rounds for each scoped campaign and for joint refinement, the latter starting from the composite. (b) Development-split AUC against held-out AUC for the 73 programs of the run that were evaluated on the held-out benchmarks, with Pearson correlation r.

E What the Agent Discovered

Section 4.3 notes that the controllers returned by CRAFT are readable programs in which two design patterns recur. Here, we describe the controller of CRAFT-B256 space by space, then the two patterns on both trackers, and close with the inference cost.

Search-region selection. Consistent direction and speed over recent frames accumulate a motion trust, and the crop is shifted ahead along the trusted motion with per-axis limits and a bounded expansion driven by the larger axis of motion. A stop or reversal resets the motion estimate and a speed mismatch quickly discounts the trust, so an unpredictable target falls back to the seed crop; while a loss is unresolved no motion is learned, and the estimate restarts after recovery.

Response decoding. The cosine window is applied additively and adaptively, so that an ambiguous response keeps the center prior whereas a strong, isolated peak relaxes it and a fast target can be followed off-center. The window is used only to choose the location; the confidence that drives template admission, loss suspicion, and the reported score is the raw response at the chosen location, so that the spatial prior does not masquerade as visual evidence.

Template-memory management. A candidate template is staged when a frame passes the admission test and stored only if the next frame passes it as well. The store is a bounded bank whose newest half is kept by recency and whose older half by quality, so that strong older appearances survive long occlusions; at each refresh, the three dynamic slots take the highest-quality entry from three equal time spans of the bank, and the first-frame template is permanently protected. A new template set is trial-run on the current search crop before it is activated and rejected if its raw confidence collapses, or drops while the peak moves or the box size changes sharply without supporting evidence. The

admission threshold is not a global constant: it is frozen once per object, within a bounded range, from the median confidence of its first trusted observations, provided those observations have stable box sizes; a template that qualifies only by this per-object threshold must additionally show a reliable, unambiguous raw peak. While a loss is unresolved, no template is learned.

Failure recovery. Loss is suspected when the raw confidence stays far below its running trusted level for consecutive frames, and a higher level is required to leave suspicion. The tracker then searches progressively wider crops around the last reliable position and, after persistent loss, scans overlapping tiles at the target's scale, ordered along the last reliable motion direction with periodic revisits of the last position. A candidate must pass score, peak-sharpness, and geometry gates, is normally confirmed by a second consistent detection, and is accepted only if, used as a template, it re-finds the target in the last reliable frame.

Recurring patterns. Two patterns recur across the accepted rounds, and neither was suggested by the prompt. First, verify before committing: new templates and recovery candidates are each tested against the current or a previous frame before they change the state, motion is trusted only after it has been consistent over several frames, and the agent repeatedly preferred a cheap verification to a stricter threshold. Second, calibrate per object rather than per dataset: the final rounds of refinement replaced global constants by quantities measured on each sequence's own trusted observations, a choice consistent with the cross-dataset transfer in Table 1. On Seed-SAM3 the same two patterns reappear in a different form. Verification guards the exit from recovery: a candidate is accepted only if it agrees across several validation views and is not contradicted by the first-frame template. Calibration is again per object rather than per dataset: the model's box is fused with the SAM3 box edge by edge, with SAM3 carrying more weight on an edge where the raw response is ambiguous or the recent boundary motion is unstable, and more weight overall when the model's confidence at the selected location is weak. The position and size checks on recovery candidates are measured in units of the target's size rather than in pixels.

Inference cost. Everything above is ordinary code around a frozen model, and its extra model calls are few: one trial run at most once per template refresh and, only after a loss, the wider crops and the tiles. The CRAFT-B256 controller runs nearly as fast as the seed (Table 4), and no language model is involved at inference.