DRIVELAB: AUTONOMOUS RESEARCH FOR SELF-DRIVING MOTION-PREDICTION MODELS

Anonymous authors
Paper under double-blind review

ABSTRACT

Developing high-performing motion-prediction models for autonomous driving is an iterative and resource-intensive process: researchers must diagnose model failures, modify complex training and inference pipelines, debug implementations, launch expensive experiments, and prioritize next steps. We investigate whether this development loop can be automated using LLM-based autonomous research agents. To this end, we build an agent-operable environment around the Waymo Open Motion Dataset (WOMD) and study the autonomous improvement of two structurally distinct motion-prediction systems: Wayformer and MotionLM. We introduce DRIVELAB, an auto-research framework that organizes long-horizon exploration through three agentic roles atop a shared human-built infrastructure: Researcher Agents that propose high-quality, diverse hypotheses; sandboxed, skill-equipped Coders that implement changes and diagnose runtime failures; and an Evolution Manager that maintains persistent experimental memory and manages context. Under a fixed 24-hour compute budget on identical hardware, DRIVELAB achieves the highest Soft Mean Average Precision (Soft mAP) among the evaluated harnesses for all three base LLMs: averaged across base LLMs, it improves Wayformer by 9.5% and MotionLM by 13.0% over their starting checkpoints (up to 18.6% without a fixed time budget), and discovers non-trivial algorithmic improvements, such as new mode-aggregation mechanisms. We also identify the balance between research-idea novelty and code-execution stability as a key design consideration for auto-research and show that our method handles both well. Our results provide strong evidence that autonomous LLM agents can take on substantial parts of the development of complex driving-behavior models.

1 INTRODUCTION

Motion prediction, defined as forecasting multiple plausible future trajectories of surrounding agents alongside their likelihoods given HD map context and historical states (Nayakanti et al., 2022; Shi et al., 2022), is critical for the safe deployment of autonomous vehicles. Yet, advancing state-of-the-art performance remains challenging. On competitive benchmarks like the Waymo Open Motion Dataset (WOMD), annual state-of-the-art gains on Soft mAP metrics have historically progressed by modest increments around 0.02 (Waymo Open Dataset, 2024). This plateau shows the inherent complexity of modern prediction pipelines: whether utilizing continuous regression over vectorized scene encoders or autoregressive trajectory decoders (Seff et al., 2023; Philion et al., 2024; Zhou et al., 2025), these architectures tightly couple scene representation, multi-agent interaction modeling, loss formulations, trajectory decoding, and ensembling or post-processing heuristics. Consequently, meaningful progress rarely stems from isolated, simple tweaks; it requires coordinated modifications across interdependent subsystems followed by compute-intensive training and human supervision.

Can the motion-prediction development loop be automated? Recent advances in long-context reasoning, agentic tool use, and multimodal scene understanding enable LLM agents to reason over large codebases, implement modifications, launch distributed experiments, and interpret complex empirical diagnostics (Yao et al., 2023; Dong et al., 2025; Cho et al., 2024; Liu et al., 2025). These capabilities raise the prospect of autonomous research systems that drive the end-to-end model development cycle: analyzing failures, proposing improvements, implementing them, and using the evidence to choose the next direction (Lyu et al., 2026).

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.

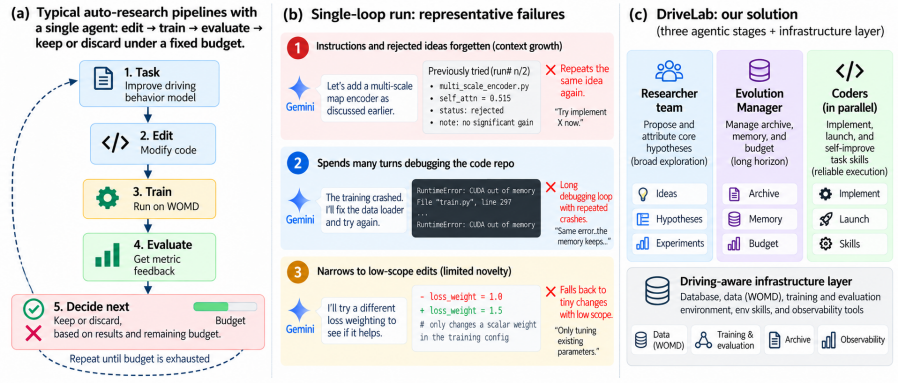

Figure 1: Overview. **(a)** Typical auto-research pipelines with a single agent: edit → train → evaluate → keep or discard under a fixed budget. **(b)** Representative failures: instructions and rejected ideas forgotten after context growth, time spent debugging the code repo, and narrowing to low-scope edits. **(c)** DRIVELAB separates Researcher, Evolution Manager, and Coders over a shared driving-aware infrastructure with persistent archive, memory, skills, and observability tools.

To study this question, we build an agent-operable environment around WOMD and two model families: Wayformer, with its non-autoregressive mixture-of-trajectory-Gaussians predictor (Nayakanti et al., 2022), and MotionLM, with its autoregressive token decoder (Seff et al., 2023). We find that harnesses based on standard edit–train–evaluate loops (Karpathy, 2026) quickly break down over long experimental horizons (Appendix E), exhibiting three major failure modes: (1) *context overload and instruction drift*, where constraints and negative results fall out of the working context, causing repeated syntax and runtime failures (Epperson et al., 2025; Wan et al., 2026); (2) *procedural rediscovery*, where the agent repeatedly relearns repository-specific debugging and evaluation conventions; and (3) *search narrowing*, where the agent exhibits failure aversion after complex edits fail, retreating into trivial updates rather than exploring substantive alternatives (Igbinedion et al., 2026). In a compute-intensive pipeline, these failures can consume accelerator time without producing new evidence, which makes auto-research less efficient and robust (Kang et al., 2026).

DRIVELAB: autoresearch harness for motion-prediction pipelines. We introduce **DRIVELAB**, designed around these failure modes to sustain scalable autonomous motion-prediction model development over long horizons. DRIVELAB splits the work across specialized agents that share persistent research state: a **Researcher team** proposes, cross-checks, and ranks diverse hypotheses across the prediction stack, balancing refinement of promising incumbents against exploration of riskier mechanisms, and dispatches them directly to sandboxed parallel **Coders**, which implement, execute, and verify them while distilling reusable procedural skills (Xu & Yan, 2026) for repository-specific workflows; an **Evolution Manager** maintains the archive and research memory, supplying each agent with bounded context (Liu et al., 2026). Each mechanism (context separation, self-distilled skills, sandboxed subagents) has precedents; our contribution is their integration and, to our knowledge, the first study of long-horizon autonomous research on motion-prediction models.

Empirical findings. We evaluate DRIVELAB on WOMD with three base LLMs, in the same environment used for the pilot study in Appendix E. Under a fixed 24-hour budget on identical hardware, DRIVELAB achieves the highest Soft mAP in every setting, improving Wayformer by 9.5% and MotionLM by 13.0% over their seed checkpoints on average across base LLMs, and by up to 18.6% in an open-ended episode. Three findings stand out. First, *the harness matters no less than the base LLM*: a stronger LLM helps under both harnesses, yet every DRIVELAB configuration outperforms every Single-Agent configuration, including one built on a stronger LLM. Second, DRIVELAB *breaks the coupling between ambition and failure*: among the baselines, more novel search comes with more failed attempts, whereas DRIVELAB achieves both the highest novelty and the lowest Unproductive Attempt Rate (UAR), keeping reserved accelerators substantially more active instead of idling while agents debug invalid patches. Third, *the agent's effort follows the objective*: gains concentrate in the pipeline stages to which Soft mAP is most sensitive, which makes the choice of objective and modifiable scope a central lever for human oversight. Case studies further

show that DRIVELAB discovers substantive, domain-adapted improvements beyond hyperparameter tuning, each traced from a diagnostic symptom to a mechanism-level fix.

2 RELATED WORK

Autonomous research agents. Recent systems increasingly automate the research loop, including hypothesis generation, implementation, experiment execution, interpretation, and iterative model improvement (Gottweis et al., 2026; Schmidgall et al., 2025; Yamada et al., 2025; Zou et al., 2026). Evaluation has correspondingly expanded from repository-level coding benchmarks (Jimenez et al., 2024) to environments for post-training, recursive self-improvement, and broader AI research (Rank et al., 2026; Chen et al., 2026; Xu et al., 2026; Wang et al., 2026; Jiang et al., 2025; Li et al., 2026), as well as scientific domains including mathematics, chemistry, robotics, and medicine (Romera-Paredes et al., 2024; Boiko et al., 2023; Wu et al., 2026b; Xiao et al., 2026). We study this problem in autonomous-driving motion prediction, where agents must improve models across tightly coupled stages such as representation, architecture, action space generation, and post-processing.

Agentic harnesses have evolved from simple edit–eval loops (Karpathy, 2026; Feng et al., 2026) to structured paradigms utilizing tree search and evolutionary archives (Wang et al., 2023; Romera-Paredes et al., 2024; Jiang et al., 2025; Yamada et al., 2025; Novikov et al., 2025), parallel and multi-agent coordination (Hu et al., 2025; Lyu et al., 2026; Wu et al., 2026a), and pre-execution screening or reflective memory to prevent long-horizon drift (Ning et al., 2026; Ji & Ding, 2026). Concurrently, efficient agent systems emphasize sandboxed execution and bounded context compression (Liu et al., 2026). Our harness integrates these principles within a role-specialized hierarchy, balancing exploratory novelty with reliable code execution under expensive experimental feedback.

Motion prediction for autonomous driving. Motion forecasting predicts future trajectories from dynamic agent states and map context (Varadarajan et al., 2021; Gu et al., 2021). We study **Wayformer** (Nayakanti et al., 2022) and **MotionLM** (Seff et al., 2023), which are representative of two of the most common contemporary paradigms for motion prediction, namely one-shot mixture of Gaussians and discrete auto-regression. Improving both tests whether autoresearch generalizes across paradigms. Recent work further advances motion prediction through intention refinement and evolving anchors (Shi et al., 2022; 2023; Lin et al., 2024), sequential mode modeling (Zhou et al., 2025), flow-based generation (Yan et al., 2025), unified prediction–planning modeling (Song et al., 2025), and robustness or adaptive context modeling (Sun et al., 2025; 2026); adding such models is left for future work. Evaluation relies on open-loop metrics such as minADE and Soft mAP. We use the Waymo Open Motion Dataset (WOMD) (Ettinger et al., 2021) as our primary benchmark because it provides large-scale, interaction-rich scenarios and standardized prediction tasks.

3 DRIVELAB: AUTO-RESEARCH PIPELINE

DRIVELAB comprises three agentic roles atop a shared human-built infrastructure (Figure 2): a Researcher team that generates and selects hypotheses, sandboxed Coders that implement and evaluate them, and an Evolution Manager that preserves accumulated evidence, maintains the parent pool and budget, and supplies bounded context across a long research episode.

Design rationale. At its core, an auto-research system is a coding agent that iteratively edits, runs code, and learns from past results. Whether an iteration yields usable evidence depends on (i) the *difficulty* of the implementation, which is set by the hypothesis; (ii) the *context* available to the agent, i.e., what it knows about the code and prior attempts and what its tools and permissions let it observe and do; and (iii) the *capability* of the base LLM. A harness controls (i) and (ii). Difficulty and payoff are not monotonically related: trivial or hyperparameter-level edits are easy to implement but yield incremental gains, whereas mechanism-level hypotheses can pay off more but fail more often. Informally, testing h under context c is worth $\Pr[\text{valid} \mid h, c] \cdot \mathbb{E}[\Delta f_{\text{dev}} \mid h, \text{valid}]$, where f is the downstream metric; a harness that only avoids failure drifts toward low-value edits, and one that only rewards ambition risks wasting compute on failed implementations.

DRIVELAB thus splits the work across three agentic roles on top of a human-built infrastructure layer (Figure 2): a **Researcher team** that decides what to test (Section 3.1), sandboxed **Coders** that raise the chance that selected hypotheses run correctly (Section 3.2), and an **Evolution Manager** that

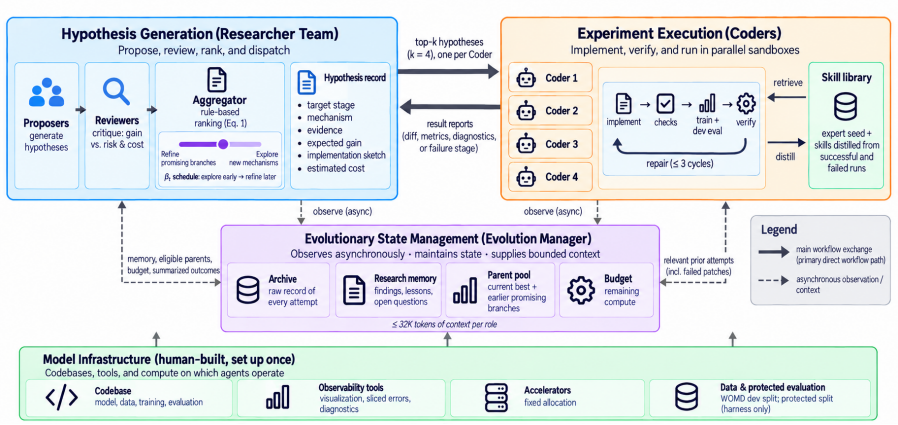

Figure 2: DRIVELAB architecture. The Researcher team proposes, critiques, and ranks hypotheses and dispatches the selected ones directly to parallel Coders in isolated sandboxes. Coders implement, test, debug, launch, and audit experiments within role-scoped permissions and report results back to the Researcher team. The Evolution Manager asynchronously records returned code diffs, metrics, diagnostics, and failure reports into a persistent archive and memory, maintains the pool of eligible parents and the budget, and supplies each role with bounded context. All roles operate over a shared human-built infrastructure with driving-specific observability tools and restrictions.

supplies bounded context from the episode's history (Section 3.3); the infrastructure supplies context from the environment through domain-specific tools and role-scoped permissions (Section 3.4). Hypotheses go directly from the Researcher team to the Coders, and results come directly back; the Manager observes this loop asynchronously and provides additional context if needed.

3.1 HYPOTHESIS GENERATION AND SELECTION

The Researcher team (Proposers, Reviewers, and a rule-based Aggregator) decides what to test next and owns the exploration–exploitation trade-off. A research state s consists of a model workspace and its development Soft mAP, $f_{\text{dev}}(s)$; a hypothesis h proposes a transformation of a parent state s_p drawn from the pool of eligible parents maintained by the Manager (Section 3.3); the archive $\mathcal{A}$ records all previous attempts with their code changes and diagnostics; and rounds are indexed by t.

Propose and review. The *Proposers* generate candidate hypotheses from the research brief, the context supplied by the Manager, diagnostics, and the remaining budget, querying the archive for related past attempts and online resources (e.g., GitHub repositories and technical reports). Each hypothesis is a structured record specifying the target pipeline stage, proposed mechanism, supporting evidence, expected gain, implementation sketch, and estimated cost. Independent *Reviewers* weigh expected gain against implementation risk and cost and output $q(h \mid s_p) \in [-1, 1]$, from likely harmful (-1) to likely beneficial $(+1)$.

Rank and dispatch. The *Aggregator* then ranks parent–hypothesis pairs deterministically by

$$\text{score}_t(s_p, h) = \underbrace{\tilde{f}_{\text{dev}}(s_p) + q(h \mid s_p)}_{\text{exploitation}} + \beta_t \underbrace{u(h, \mathcal{A})}_{\text{exploration}}, \tag{1}$$

where $\tilde{f}_{\text{dev}}(s_p) \in [0, 1]$ is the parent's Soft mAP, min–max normalized over archived states with a valid development score (best so far $= 1$, worst $= 0$), placing it on a scale comparable to q. The exploration term, or archive dissimilarity, is $u(h, \mathcal{A}) = 1 - \max_{e \in \mathcal{A}} \text{sim}(h, e) \in [0, 1]$, where sim is a deterministic, normalized token-edit similarity between the candidate's canonical edit signature and each archived successful or failed patch; no learned encoder or LLM judge is used (Appendix D). We set $u(h, \emptyset) = 1$; when the valid state-score range is zero, we set $\tilde{f}_{\text{dev}} = 1$ for the eligible parent; and failed attempts without a valid f_{dev} are excluded from state-score normalization. Consequently, retries targeting the same parts of the code receive $u \approx 0$. The exploration weight follows the schedule $\beta_t = \beta_0 \gamma^t$ with $\beta_0 = 1.5$ and $\gamma = 0.92$, so early rounds favor diverse mechanisms and later

rounds favor refining promising branches. The Aggregator dispatches the top-k pairs ($k = 4$) whose estimated cost fits the remaining budget directly to parallel Coders, one pair per Coder. All terms are ranking heuristics rather than calibrated predictions, and u is distinct from the reported *Novelty* metric (Section 4.1), which is computed post hoc by a judge panel and never exposed to the loop.

3.2 Experiment Execution

Implement, debug, and verify. A Coder is a sandboxed repository-level agent responsible for implementing and validating one dispatched pair (s_p, h). It receives the hypothesis from the Researcher team, together with the relevant prior attempts attached by the Manager (Section 3.3). Starting from an isolated checkout of the parent state, it inspects the relevant code and configuration, implements the hypothesis, and verifies that the resulting candidate behaves as intended. Before launching a full experiment, the Coder performs inexpensive checks such as configuration validation, unit tests, and small-shard smoke tests. It then launches training, within its training budget (Appendix D), and development evaluation through the managed job interface, and monitors logs, checkpoints, metrics, and job health. When execution fails, the Coder diagnoses the available artifacts, repairs the implementation, and reruns the appropriate checks, with at most three repair cycles per hypothesis. A candidate is accepted only after the Coder verifies that the intended code and configuration were executed and that the expected artifacts and metrics were produced. The Coder then reports directly back to the Researcher team: successful attempts return the validated code diff, configuration, metrics, diagnostics, and implementation report; unsuccessful attempts return the failure stage and attempted repairs. The Manager records the full returned state (Section 3.3). Up to $k = 4$ Coders run in **parallel in isolated sandboxes**. Each Coder shares only read-only infrastructure and the skill library and cannot inspect or modify sibling workspaces, allowing independent implementation and debugging without interfering with other branches.

Seed and automatically distilled skills. Research agents repeatedly encounter repository-specific procedures, such as launch conventions or export constraints, that are difficult to infer reliably from source code alone. Coders therefore share a skill library in the Agent Skills (`SKILL.md`) format, initialized with expert-authored seed workflows and expanded automatically during the episode (Xu & Yan, 2026). Whenever a Coder needs more than ten steps to complete a subtask, a distillation step runs after the subtask finishes and extracts reusable implementation knowledge: preconditions, coding conventions, and recurring error patterns with their fixes. Skills are indexed by model family and pipeline stage and retrieved by later Coders when relevant, reducing repeated debugging and rediscovery across experiments. Appendix F labels the provenance of each representative skill.

3.3 Evolutionary State Management

The Evolution Manager maintains the episode's evolving population of research states, but it sits beside the Researcher–Coder loop: it does not spawn agents, select or dispatch hypotheses, or balance exploration against exploitation. It asynchronously observes every dispatched hypothesis and returned result, maintains the archive, the research memory, and the pool of eligible parents, tracks the remaining budget, and supplies the other agents with bounded context drawn from this state.

Archive, memory, and parents. The Manager stores the complete raw state of each attempt in $\mathcal{A}$, including the hypothesis and its rationale, parent state, code changes, execution logs, metrics, diagnostics, and failure reports. It also maintains a compact research memory of **findings**, **lessons**, and **open questions**, consolidated after every four completed experiments. The pool of eligible parents includes the current best state but also earlier promising branches; a candidate is promoted only if its development Soft mAP improves on its parent's by at least $\tau = 0.005$ (Section 4.1).

Bounded, role-specific context. Rather than exposing the growing history directly to every agent, the Manager supplies each role with bounded context. Each Researcher round receives the research memory, the pool of eligible parents, the remaining budget, and the latest outcomes summarized as evidence, and the Proposers can query the archive for related past attempts. Each Coder receives, alongside its hypothesis, the relevant prior attempts, including failed patches and their failure reports. Together with retrieved archive entries, such context is capped at 32K tokens per role. The cap excludes system prompts, the current task, code, and live tool interactions, so it does not bound the total live context. Information is thus repeatedly compressed for communication but never discarded, and the cycle continues until the budget is exhausted or the stopping criterion is reached.

3.4 Model Infrastructure and Domain Observability

The infrastructure layer makes two motion-prediction families operable by research agents under a common evaluation protocol. The Wayformer pipeline uses scene-centric transformer representations with a one-shot mixture of Gaussians predictor (Nayakanti et al., 2022), while the MotionLM pipeline discretizes motion and autoregressively generates future trajectories (Seff et al., 2023). Both expose the research-relevant stages of preprocessing, model definition, training, and sampling/post-processing. Each candidate is trained and evaluated as a single model rather than an ensemble.

Driving-specific tools and observability. All methods receive the same repository and interfaces, together with read-only tools for scene visualization, sliced error analysis, failure-case retrieval, training-curve inspection, and parent–candidate comparison. These tools expose visual outputs and compact structured diagnostics such as per-mode displacement, heading error, off-road behavior, mode spread, and failure categories, helping agents connect model failures to specific parts of the pipeline before modifying a mechanism. More details are provided in Appendix B.

Restrictions and permissions. Capabilities are partitioned across roles to ensure consistency and context management. The Researcher team selects and dispatches what should be tested but cannot edit code or execute experiments. The Coder determines how an assigned hypothesis is implemented and validated: it can modify its sandboxed repository, run tests and smoke tests, launch and monitor experiments, diagnose failures, repair implementations, and inspect development metrics, but cannot select future research branches, modify persistent research state, inspect sibling workspaces, or invoke protected evaluation. The Evolution Manager maintains the archive, memory, pool of eligible parents, and budget, but cannot modify code, launch jobs, or select hypotheses. A subset of the evaluation set (protected split in Section 4.1) is never mounted in agent sandboxes; only the harness can invoke protected evaluation after an episode terminates, and protected scores are never returned to the agents or used to resume search. These restrictions are enforced by the scaffold's permission hooks and sandbox layers rather than prompting (Table 5).

4 Experiments

4.1 Evaluation Protocol

Task and baselines. We evaluate on the Waymo Open Motion Dataset (WOMD) marginal prediction task (Ettinger et al., 2021). We compare DriveLab with three automated research harnesses under identical 24-hour compute budgets and evaluation interfaces: **Single-Agent**, a sequential edit–train–evaluate harness based on the AutoResearch recipe (Karpathy, 2026); **Role-based MAS**, a Planner–Reviewer–Coder system following recent multi-agent research frameworks (Hong et al., 2024; Hu et al., 2025; Lyu et al., 2026); and **Antigravity Scaffold**, a baseline using Google Antigravity's `/goal` autonomous-execution workflow (Google, 2026). We evaluate Gemini 3.6 Flash (Google DeepMind, 2026b), Claude Opus 5 (Anthropic, 2026), and Gemini 3.1 Pro (Google DeepMind, 2026a) as base LLMs. More details on baselines are presented in Appendix D.

Metrics. Our primary model-quality metric is *Soft mAP*, the official ranking metric of the Waymo Open Motion Dataset benchmark. It evaluates top-K multimodal trajectory predictions using a velocity-dependent spatial hit window, scores the single highest-confidence match per ground-truth trajectory, and ignores additional matching predictions. To characterize the research process, we additionally report *Novelty*, *Unproductive Attempt Rate* (UAR), and research efficiency. Novelty (0–3) is scored post hoc on every dispatched attempt by a two-judge panel from the diff alone. Scope is computed deterministically for the Single-Agent diagnostic analysis and case studies (Appendix A). UAR is the fraction of dispatched attempts that fail to produce a valid experiment; valid but non-improving experiments are not counted as unproductive. To measure execution efficiency, we report *LLM tokens per valid experiment* and the *Active Utilization Rate*. The former captures how much agent inference is required to produce one successfully executed and evaluated experiment. The latter is one minus the aggregate idle-time ratio of all reserved accelerator hardware over the full research window, so higher values indicate that more reserved hardware capacity was actively used. Full metric definitions and evaluation details are provided in Appendix A.

Evaluation. All models use the official WOMD training split. We partition the 150 validation shards into 80 *development* shards used throughout research and 70 *protected* shards that remain

Table 1: **Fixed 24-hour auto-research experiments on WOMD.** Soft mAP denotes the protected split score of the development selected final checkpoint, reported as the **mean [min, max]** across five independent episodes. Research Process metrics are averaged over the two model families. See Appendix D for additional metric results (e.g., minADE).

		Soft mAP ↑ mean [min, max]		Research Process			
Base LLM	**Method**	**Wayformer**	**MotionLM**	**Novelty ↑**	**UAR (%) ↓**	**Tok/exp (M) ↓**	**AUR (%) ↑**
Initial seed checkpoint		0.3788	0.3905	–	–	–	–
Gemini 3.6 Flash	Single-Agent	0.3855 [0.3788, 0.3913]	0.4037 [0.3905, 0.4159]	0.55	22.0	1.30	58.4
	Role-based MAS	0.3951 [0.3788, 0.4045]	0.4118 [0.3905, 0.4243]	0.89	28.9	0.67	62.5
	Antigravity Scaffold	0.3998 [0.3957, 0.4020]	0.4146 [0.4009, 0.4233]	0.94	35.5	0.80	53.0
	DRIVELAB (ours)	**0.4128** [0.4041, 0.4255]	**0.4432** [0.4303, 0.4516]	**1.35**	**9.1**	**0.55**	**80.5**
Claude Opus 5	Single-Agent	0.3875 [0.3825, 0.3930]	0.4228 [0.4115, 0.4332]	0.62	12.5	1.08	63.5
	DRIVELAB (ours)	**0.4188** [0.4100, 0.4284]	**0.4515** [0.4450, 0.4553]	**1.28**	**3.8**	**0.39**	**85.7**
Gemini 3.1 Pro	Single-Agent	0.3830 [0.3819, 0.3850]	0.4015 [0.3989, 0.4033]	0.45	18.8	1.16	53.9
	DRIVELAB (ours)	**0.4125** [0.4027, 0.4223]	**0.4296** [0.4205, 0.4377]	**1.21**	**10.7**	**0.51**	**83.1**

inaccessible to all agents until the episode terminates. Candidate evaluation, promotion, and plateau detection use only the development split. Once search is frozen, the final development-selected incumbent is evaluated on the protected split. In the primary setting, all methods start from the same codebase and receive 24 hours with a fixed reservation of four accelerators. Candidate promotion requires a development Soft mAP improvement of at least the prespecified conservative repeatability threshold $\tau_{\text{promote}} = 0.005$. We additionally run single-episode open-ended studies that terminate after 10 consecutive valid non-improving evaluations or 12 hours without improvement.

4.2 MAIN RESULTS

Open-ended results: baselines stall, DRIVELAB keeps improving. Without a fixed budget, the baseline incumbents stop improving within the first day and meet the plateau criterion, whereas DRIVELAB continues to find promotable candidates beyond the 24-hour mark (Figure 3), running for 61.6 hours before plateauing (Table 2). Its final incumbent reaches 0.4632 Soft mAP on the protected eval set, approximately 18.6% above the seed. As this is a single MotionLM episode, we read it as evidence that DRIVELAB had not exhausted useful hypotheses at 24 hours, not as a general scaling law.

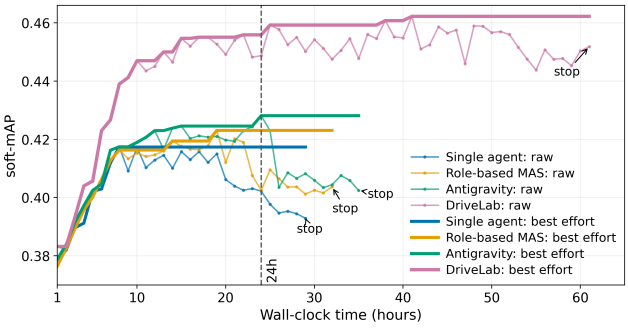

Figure 3: **Open-ended auto-research evaluated with MotionLM/Gemini 3.6 Flash.** Markers connected by thin lines denote Soft mAP and may drop when a new candidate underperforms; thick lines trace the best-so-far score.

Fixed-time results: both the base LLM and the harness matter. Table 1 compares all four harnesses with Gemini 3.6 Flash, and DRIVELAB against Single-Agent for the other two base LLMs. Claude Opus 5 yields the strongest results under both harnesses, but the harness matters no less: the smallest gain from switching to DRIVELAB exceeds the largest difference between base LLMs, and the weakest DRIVELAB configuration outperforms the strongest Single-Agent one. The gains are robust to run-to-run variation: in five of six settings, DRIVELAB's *worst* episode exceeds every baseline's *best* (the exception, Wayformer with Gemini 3.6 Flash, trails by 0.0004). With Gemini 3.6 Flash, its MotionLM gain over the seed is about 4× that of Single-Agent and 2.2× that of the strongest alternative. For context, year-over-year gains among top single-model WOMD entries are typically around 0.02 (Waymo Open Dataset, 2024); our seeds start below the state of the art, so the gains are not directly comparable (Appendix D), but they show that auto-research can deliver meaningful improvements within a day.

Novelty, reliability, and efficiency: breaking the ambition–failure coupling. Among the baselines (Gemini 3.6 Flash), higher Novelty comes with higher UAR (Single-Agent → Role-based

Table 2: Component ablations (Gemini 3.6 Flash, MotionLM, open-ended evaluation, one episode each). Soft mAP denotes the protected split score of the final checkpoint. Hours: wall-clock time until the open-ended stopping rule is met. Tok/exp: LLM tokens (M) per valid experiment.

Configuration	Soft mAP ↑	Novelty ↑	UAR (%) ↓	Tok/exp (M) ↓	Hours
DRIVELAB (full)	**0.4632**	**1.48**	10.5	0.52	61.6
− Researcher Team	0.4197	0.58	14.2	0.61	31.8
− Skill Distillation	0.4350	1.25	27.6	0.98	49.8
− Exploration term ($\beta = 0$)	0.4376	0.75	**5.7**	**0.41**	55.4

MAS → Antigravity: $0.55 \rightarrow 0.89 \rightarrow 0.94$ and $22.0\% \rightarrow 28.9\% \rightarrow 35.5\%$). DRIVELAB breaks this coupling, achieving the highest Novelty and the lowest UAR for every base LLM: *it not only proposes more ambitious ideas but also makes them executable.* This reliability keeps accelerators busy instead of idle during repeated repairs: DRIVELAB uses 80–86% of the reserved hardware versus 53–64% for the baselines, with less than half the tokens per valid experiment of Single-Agent. The full system avoids much of the repeated debugging documented in Appendix E; removing skills alone more than doubles UAR (Table 2).

4.3 ADDITIONAL ANALYSIS

Component analysis: good ideas and reliable execution are both necessary. Removing any of the three evaluated components lowers the final Soft mAP by at least 0.025, but each removal fails in a different way (Table 2). *Idea generation.* In −Researcher Team, planning is delegated to the Coders, which propose and implement hypotheses directly; with $\beta = 0$, the Aggregator ranks hypotheses by exploitation alone. In both cases, Novelty drops sharply and the search drifts toward incremental edits. Such edits are easy to implement, and with $\beta = 0$ both UAR and token cost per experiment are the lowest of all configurations. However, incremental edits rarely yield large gains, so both runs finish well below the full system. *Reliable execution.* In −Skill Distillation, the Coders keep only the seed skills and no longer distill reusable workflows during the episode, so they cannot improve themselves. They still propose ambitious ideas, with Novelty close to that of the full system, but they repeatedly rediscover related workflows: more attempts fail, each valid experiment costs nearly twice as many tokens, and much of the long run goes to debugging rather than producing new evidence. Only the full system combines novel ideas with reliable, efficient execution.

Table 3: ΔSoft mAP under restricted optimization scope (Gemini 3.6 Flash). **All rows are open-ended runs** terminated by the plateau criterion; Deltas use the Wayformer seed 0.3788 and MotionLM seed 0.3905.

Family	Allowed scope	Single-Agent		DRIVELAB (ours)	
		ΔSoft mAP ↑	UAR (%) ↓	ΔSoft mAP ↑	UAR (%) ↓
Wayformer	Post-processing only	+0.0151	21.2	+0.0403	8.2
	Training only	+0.0175	25.5	+0.0209	11.4
	Full loop (co-opt)	+0.0197	28.0	+0.0500	15.2
MotionLM	Post-processing only	+0.0180	22.8	+0.0572	10.0
	Training only	+0.0206	24.3	+0.0327	12.5
	Full loop (co-opt)	+0.0232	29.5	**+0.0727**	10.5

Optimization scope: effort follows the objective. Beyond role permissions (Section 3.4), the environment lets the human restrict which pipeline stages the agents may modify. We compare three scopes (Table 3): post-processing only, which edits sampling, mode aggregation, and confidence scoring on a fixed trained model; training only, which edits the stages that require retraining while post-processing stays fixed; and full loop, where the agent may modify both and decides on each attempt where to spend its effort (which is the default setting in the main experiments). Post-processing alone already yields most of the gain: without retraining, DRIVELAB improves Wayformer by +0.0403 and MotionLM by +0.0572, about 80% of the full-loop gain. This is expected, since Soft mAP scores only the highest-confidence match per ground-truth trajectory and ranks predictions by confidence, making it highly sensitive to mode selection and scoring. DRIVELAB leads Single-Agent under every scope (e.g., 3.2× on MotionLM with post-processing only), and the full loop remains best (+0.0500 and +0.0727), though below the sum of the stage-wise gains. **Scope restriction thus lets a domain expert focus the agents on the stages the objective is most sensitive to**, capturing most of the gain with cheaper attempts, or exclude stages whose metric gains may not reflect better predictions.

4.4 CASE STUDIES: AUTOMATED PATTERN DISCOVERY

Beyond aggregate performance, we examine two discoveries (Appendix G for details) in which DRIVELAB proposes mechanism-level fixes rather than hyperparameter adjustments through long-horizon autoresearch.

Case 1: Hybrid discrete–continuous residual refinement. MotionLM represents motion with a $13 \times 13 = 169$ grid of quantized Verlet actions, whose center token denotes zero acceleration. After several research rounds, the Researcher observed that minADE had plateaued while sequence cross-entropy continued to improve and attributed this decoupling to action-space quantization. Rather than increasing vocabulary resolution and decoding cost, it proposed retaining discrete tokens for multimodal branching while adding a lightweight head that predicts continuous sub-grid residuals in Verlet-action space from the shared decoder embeddings. The corrected action is then integrated by the Verlet decoder to obtain the trajectory. The Coder implemented the residual head and jointly optimized a Smooth-L_1 refinement loss with the autoregressive objective. This modifies both the output representation and the training objective while preserving MotionLM's discrete branching structure. On the protected split, the resulting model improves **Soft mAP from** 0.3905 **to** 0.4284.

Case 2: Multi-horizon diversity-aware mode aggregation. Diagnostics throughout auto-research revealed two failures in the baseline aggregator: K-means pulls distinct modes toward cluster interiors, and single-horizon ($5.0\,\mathrm{s}$) suppression prunes modes that branch late. DRIVELAB proposed multi-horizon NMS, with three components. First, pairwise distances between the $N = 512$ rollouts are computed at 3.0, 5.0, and $8.0\,\mathrm{s}$, normalized by speed-dependent radii, and combined in a weighted score. Rollouts are treated as duplicates when this aggregate score is at most one; the rule emphasizes the $5.0\,\mathrm{s}$ horizon but does not require closeness at every horizon. Second, the $K = 6$ modes are selected by Maximal Marginal Relevance (MMR), trading off covered probability mass against distance from already-selected modes. Third, K-means refinement is replaced by entropic optimal transport: rollouts are softly assigned to modes, and each mode is refined as its transport-weighted barycenter, blended with the original rollout by a speed-dependent weight that retains a bounded seed contribution (Scope 2, Novelty 2; see Appendix G.2). On the protected split, it raises Soft mAP to 0.4473 with the post-processing change alone and to 0.4551 when combined with the Case 1 refinement head. With hyperparameters tuned on the development split, it also **achieves 0.4523 Soft mAP on the official WOMD test benchmark without ensembling or LiDAR inputs (Appendix D)**, supporting generalization beyond our protected split.

5 CONCLUSION

We frame autonomous research on motion-prediction models as a long-horizon systems problem and introduce DRIVELAB, which includes hypothesis generation with balanced exploration and exploitation, self-improving coding agents, and persistent evolutionary state. Our findings suggest that in domains like motion prediction where experiments are expensive, the bottleneck shifts from generating ideas to organizing research: what to remember, how implementation knowledge accumulates, and how to balance ambition against failure risk. This implies, first, that the research harness deserves study as a first-class object, evaluated by process measures such as unproductive attempts and hardware utilization, not only final scores. Second, because an autonomous researcher concentrates effort wherever the objective is most sensitive, choosing what to optimize and what the agent may modify becomes the primary lever of human oversight.

Future Work and Limitations. DRIVELAB currently improves an offline behavior predictor. A natural next step is closed-loop optimization of planning or control components in simulation, with a protected simulator and scenario suite replacing the offline evaluator. There, specifying the objective and protecting its evaluation will matter as much as agent capability. Extending the loop to runtime decision-making would also require stronger safety and validation guarantees. We also acknowledge that making a model stack agent-operable requires one-time manual infrastructure work. Ablation and scope analyses use single open-ended episodes due to their compute cost, so small differences between variants should be interpreted cautiously. Finally, open-loop WOMD metrics are not evidence of deployment safety; any extension to deployed systems should retain protected evaluation and human review of the proposed changes.

AI USE STATEMENT

We use generative AI as the subject of study: the evaluated LLM agents generate hypotheses and code changes. Agent-generated code is re-evaluated by the harness and manually inspected for reported case studies; research changes presented as novel discoveries are manually checked against the literature. We also used AI to help refine the language of the paper and the design of figures. The authors take responsibility for the final paper.

REPRODUCIBILITY STATEMENT

Section 3 describes the harness architecture and control flow. The appendices provide the metric definitions and token and accelerator-utilization accounting (Appendix A); role permissions and tool interfaces (Appendix B); a consolidated harness specification (Appendix C); the experimental setup (Appendix D); an analysis of Single-Agent failure modes (Appendix E); representative agent skills (Appendix F); and case-study methodology and implementations (Appendix G). Section 4.1 and Appendix D specify the evaluation protocol, including the deterministic partition of the official WOMD validation set into 80 development and 70 protected shards, compute budgets, the promotion threshold, stopping rules, baselines, seed checkpoints, hyperparameters, and the hypothesis-similarity procedure. All experiments use the public WOMD v1.2.0 dataset and the named, publicly documented proprietary base LLMs Gemini 3.6 Flash, Claude Opus 5, and Gemini 3.1 Pro. Because agent trajectories are stochastic and hosted proprietary models may change over time, Table 1 reports five independent episodes per primary setting rather than a single run. After each episode terminates, the harness evaluates the development-selected incumbent once on the protected split and performs the reported post-hoc protected evaluations of case-study components (Section 4.4); protected results are never returned to agents or used to resume search. The external test-set result in Appendix D was obtained from the official WOMD evaluation server.

REFERENCES

Anthropic. Claude Opus 5 system card. Online, July 2026. URL https://www.anthropic.com/claude-opus-5-system-card. Accessed: 2026-09-16.

Daniil A. Boiko, Robert MacKnight, Ben Kline, and Gabe Gomes. Autonomous chemical research with large language models. *Nature*, 624(7992):570–578, 2023. doi: 10.1038/s41586-023-06792-0. URL https://doi.org/10.1038/s41586-023-06792-0.

Mingguang Chen, Licheng Wang, and Bo Qu. Recursive self-improvement in AI: From bounded self-refinement to autonomous research loops, 2026. URL https://arxiv.org/abs/2607.07663.

Jang Hyun Cho, Boris Ivanovic, Yulong Cao, Edward Schmerling, Yue Wang, Xinshuo Weng, Boyi Li, Yurong You, Philipp Krähenbühl, Yan Wang, et al. Language-image models with 3D understanding. *arXiv preprint arXiv:2405.03685*, 2024. URL https://arxiv.org/abs/2405.03685.

Yihong Dong, Xue Jiang, Jiaru Qian, Tian Wang, Kechi Zhang, Zhi Jin, and Ge Li. A survey on code generation with LLM-based agents, 2025. URL https://arxiv.org/abs/2508.00083.

Will Epperson, Gagan Bansal, Victor C Dibia, Adam Fourney, Jack Gerrits, Erkang (Eric) Zhu, and Saleema Amershi. Interactive debugging and steering of multi-agent AI systems. In *Proceedings of the 2025 CHI Conference on Human Factors in Computing Systems*, CHI '25, New York, NY, USA, 2025. Association for Computing Machinery. ISBN 9798400713941. doi: 10.1145/3706598.3713581. URL https://doi.org/10.1145/3706598.3713581.

Scott Ettinger, Shuyang Cheng, Benjamin Caine, Chenxi Liu, Hang Zhao, Sabeek Pradhan, Yuning Chai, Ben Sapp, Charles R. Qi, Yin Zhou, Zoey Yang, Aurélien Chouard, Pei Sun, Jiquan Ngiam, Vijay Vasudevan, Alexander McCauley, Jonathon Shlens, and Dragomir Anguelov. Large scale interactive motion forecasting for autonomous driving: The Waymo Open Motion Dataset. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, pp. 9710–9719, October 2021. URL https://openaccess.thecvf.com/content/ICCV2021/

html/Ettinger_Large_Scale_Interactive_Motion_Forecasting_for_Autonomous_Driving_The_Waymo_ICCV_2021_paper.html.

Shiyang Feng, Runmin Ma, Xiangchao Yan, Yue Fan, Yusong Hu, Songtao Huang, Shuaiyu Zhang, Zongsheng Cao, Tianshuo Peng, Jiakang Yuan, Zijie Guo, Zhijie Zhong, Shangheng Du, Weida Wang, Jinxin Shi, Yuhao Zhou, et al. InternAgent-1.5: A unified agentic framework for long-horizon autonomous scientific discovery, 2026. URL https://arxiv.org/abs/2602.08990.

Google. Antigravity 2.0. Online, 2026. URL https://antigravity.google/docs/overview/. Accessed: 2026-09-16.

Google DeepMind. Gemini 3.1 Pro model card. Online, February 2026a. URL https://deepmind.google/models/model-cards/gemini-3-1-pro/. Accessed: 2026-09-16.

Google DeepMind. Gemini 3.6 Flash model card. Online, July 2026b. URL https://deepmind.google/models/model-cards/gemini-3-6-flash/. Accessed: 2026-09-16.

J. Gottweis, W. H. Weng, A. Daryin, et al. Accelerating scientific discovery with Co-Scientist. *Nature*, 655:487–496, 2026. doi: 10.1038/s41586-026-10644-y. URL https://doi.org/10.1038/s41586-026-10644-y.

Junru Gu, Chen Sun, and Hang Zhao. DenseTNT: End-to-end trajectory prediction from dense goal sets, 2021. URL https://arxiv.org/abs/2108.09640.

Sirui Hong, Mingchen Zhuge, Jonathan Chen, Xiawu Zheng, Yuheng Cheng, Ceyao Zhang, Jinlin Wang, Zili Wang, Steven Ka Shing Yau, Zijuan Lin, Liyang Zhou, Chenyu Ran, Lingfeng Xiao, Chenglin Wu, and Jürgen Schmidhuber. MetaGPT: Meta programming for a multi-agent collaborative framework. In *The Twelfth International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=VtmBAGCN7o.

Mengkang Hu, Yuhang Zhou, Wendong Fan, Yuzhou Nie, Bowei Xia, Tao Sun, Ziyu Ye, Zhaoxuan Jin, Yingru Li, Qiguang Chen, Zeyu Zhang, Yifeng Wang, Qianshuo Ye, Bernard Ghanem, Ping Luo, and Guohao Li. OWL: Optimized workforce learning for general multi-agent assistance in real-world task automation. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*, 2025. URL https://openreview.net/forum?id=MBJ46gd1CT.

Ifueko Igbinedion, Jillian Ross, Etienne Ricardez, Sertac Karaman, and Eric So. Lost in context: Addressing context anxiety in large language models. In *Forty-third International Conference on Machine Learning*, 2026. URL https://openreview.net/forum?id=qhbC2jtmVW.

Jiazhen Ji and Shouhong Ding. Rehearse: Stepping back from the confidence cliff in self-improving autoresearch, 2026. URL https://arxiv.org/abs/2607.27687.

Zhengyao Jiang, Dominik Schmidt, Dhruv Srikanth, Dixing Xu, Ian Kaplan, Deniss Jacenko, and Yuxiang Wu. AIDE: AI-driven exploration in the space of code, 2025. URL https://arxiv.org/abs/2502.13138.

Carlos E Jimenez, John Yang, Alexander Wettig, Shunyu Yao, Kexin Pei, Ofir Press, and Karthik R Narasimhan. SWE-bench: Can language models resolve real-world GitHub issues? In *The Twelfth International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=VTF8yNQM66.

Minki Kang, Wei-Ning Chen, Dongge Han, Huseyin A. Inan, Lukas Wutschitz, Yanzhi Chen, Robert Sim, and Saravan Rajmohan. ACON: Optimizing context compression for long-horizon LLM agents. In *Proceedings of the 43rd International Conference on Machine Learning*, 2026. URL https://arxiv.org/abs/2510.00615.

Andrej Karpathy. autoresearch: AI agents running research on single-GPU nanochat training automatically. GitHub repository, 2026. URL https://github.com/karpathy/autoresearch.

Yu Li, Chenyang Shao, Xinyang Liu, Ruotong Zhao, Peijie Liu, Hongyuan Su, Zhibin Chen, Qinglong Yang, Anjie Xu, Yi Fang, Qingbin Zeng, Tianxing Li, Jingbo Xu, Fengli Xu, Yong Li, and Tie-Yan Liu. AutoSOTA: An end-to-end automated research system for state-of-the-art AI model discovery, 2026. URL https://arxiv.org/abs/2604.05550.

Longzhong Lin, Xuewu Lin, Tianwei Lin, Lichao Huang, Rong Xiong, and Yue Wang. EDA: Evolving and distinct anchors for multimodal motion prediction. *Proceedings of the AAAI Conference on Artificial Intelligence*, 38(4):3432–3440, March 2024. doi: 10.1609/aaai.v38i4.28130. URL https://ojs.aaai.org/index.php/AAAI/article/view/28130.

Haozhe Liu, Tian Ye, Sensen Gao, Qihang Cao, Yitong Li, Mingchen Zhuge, Duomin Wang, Ruihua Zhang, Ping Luo, Jiawang Bian, Lei Zhu, Ligeng Zhu, Enze Xie, and Song Han. SoL-Pi: Recursively scaling auto-research loops for efficient agent harness, 2026. URL https://arxiv.org/abs/2609.20519.

Jiaheng Liu, Dawei Zhu, Zhiqi Bai, Yancheng He, Huanxuan Liao, Haoran Que, Zekun Wang, Chenchen Zhang, Ge Zhang, Jiebin Zhang, Yuanxing Zhang, Zhuo Chen, Hangyu Guo, Shilong Li, Ziqiang Liu, Yong Shan, Yifan Song, Jiayi Tian, Wenhao Wu, Zhejian Zhou, Ruijie Zhu, Junlan Feng, Yang Gao, Shizhu He, Zhoujun Li, Tianyu Liu, Fanyu Meng, Wenbo Su, Yingshui Tan, Zili Wang, Jian Yang, Wei Ye, Bo Zheng, Wangchunshu Zhou, Wenhao Huang, Sujian Li, and Zhaoxiang Zhang. A comprehensive survey on long context language modeling, 2025. URL https://arxiv.org/abs/2503.17407.

Yougang Lyu, Xi Zhang, Xinhao Yi, Yuyue Zhao, Shuyu Guo, Wenxiang Hu, Jan Piotrowski, Jakub Kaliski, Jacopo Urbani, Zaiqiao Meng, Lun Zhou, and Xiaohui Yan. EvoScientist: Towards multi-agent evolving AI scientists for end-to-end scientific discovery. *arXiv preprint arXiv:2603.08127*, 2026. URL https://arxiv.org/abs/2603.08127.

Nigamaa Nayakanti, Rami Al-Rfou, Aurick Zhou, Kratarth Goel, Khaled S. Refaat, and Benjamin Sapp. Wayformer: Motion forecasting via simple & efficient attention networks, 2022. URL https://arxiv.org/abs/2207.05844.

Xuying Ning, Katherine Tieu, Dongqi Fu, Tianxin Wei, Zihao Li, Yuanchen Bei, Jiaru Zou, Mengting Ai, Zhining Liu, Ting-Wei Li, Lingjie Chen, Yanjun Zhao, Ke Yang, Bingxuan Li, Cheng Qian, Gaotang Li, Xiao Lin, Zhichen Zeng, Ruizhong Qiu, Sirui Chen, Yifan Sun, Xiyuan Yang, Ruida Wang, Rui Pan, Chenyuan Yang, Dylan Zhang, Liri Fang, Zikun Cui, Yang Cao, Pan Chen, Dorothy Sun, Ren Chen, Mahesh Srinivasan, Nipun Mathur, Yinglong Xia, Hong Li, Hong Yan, Pan Lu, Lingming Zhang, Tong Zhang, Hanghang Tong, and Jingrui He. Code as agent harness, 2026. URL https://arxiv.org/abs/2605.18747.

Alexander Novikov, Ngân Vũ, Marvin Eisenberger, Emilien Dupont, Po-Sen Huang, Adam Zsolt Wagner, Sergey Shirobokov, Borislav Kozlovskii, Francisco J. R. Ruiz, Abbas Mehrabian, M. Pawan Kumar, Abigail See, Swarat Chaudhuri, George Holland, Alex Davies, Sebastian Nowozin, Pushmeet Kohli, and Matej Balog. AlphaEvolve: A coding agent for scientific and algorithmic discovery, 2025. URL https://arxiv.org/abs/2506.13131.

Jonah Philion, Xue Bin Peng, and Sanja Fidler. Trajeglish: Traffic modeling as next-token prediction, 2024. URL https://arxiv.org/abs/2312.04535.

Ben Rank, Hardik Bhatnagar, Ameya Prabhu, Shira Eisenberg, Karina Nguyen, Matthias Bethge, and Maksym Andriushchenko. PostTrainBench: Can LLM agents automate LLM post-training? In *International Conference on Machine Learning (ICML)*, 2026. URL https://arxiv.org/abs/2603.08640.

Bernardino Romera-Paredes, Mohammadamin Barekatain, Alexander Novikov, Matej Balog, et al. Mathematical discoveries from program search with large language models. *Nature*, 625(7995): 468–475, 2024. doi: 10.1038/s41586-023-06924-6. URL https://doi.org/10.1038/s41586-023-06924-6.

Samuel Schmidgall, Yusheng Su, Ze Wang, Ximeng Sun, Jialian Wu, Xiaodong Yu, Jiang Liu, Michael Moor, Zicheng Liu, and Emad Barsoum. Agent Laboratory: Using LLM agents as research assistants. *arXiv*, 2025. doi: 10.48550/arxiv.2501.04227. URL https://arxiv.org/abs/2501.04227.

Ari Seff, Brian Cera, Dian Chen, Mason Ng, Aurick Zhou, Nigamaa Nayakanti, Khaled S. Refaat, Rami Al-Rfou, and Benjamin Sapp. MotionLM: Multi-agent motion forecasting as language modeling. In *2023 IEEE/CVF International Conference on Computer Vision (ICCV)*, pp. 8545–8556, 2023. doi: 10.1109/ICCV51070.2023.00788. URL https://doi.org/10.1109/ICCV51070.2023.00788.

Shaoshuai Shi, Li Jiang, Dengxin Dai, and Bernt Schiele. MTR-A: 1st place solution for 2022 Waymo Open Dataset challenge–motion prediction. *arXiv preprint arXiv:2209.10033*, 2022. URL https://arxiv.org/abs/2209.10033.

Shaoshuai Shi, Li Jiang, Dengxin Dai, and Bernt Schiele. MTR++: Multi-agent motion prediction with symmetric scene modeling and guided intention querying. *arXiv preprint arXiv:2306.17770*, 2023. URL https://arxiv.org/abs/2306.17770.

Nan Song, Junzhe Jiang, Jingyu Li, Xiatian Zhu, and Li Zhang. UniMotion: A unified motion framework for simulation, prediction and planning. In *Advances in Neural Information Processing Systems*, volume 38, pp. 135052–135074, 2025. doi: 10.52202/085713-4063. URL https://doi.org/10.52202/085713-4063.

Jiawei Sun, Jiahui Li, Tingchen Liu, Chengran Yuan, Shuo Sun, Zefan Huang, Anthony Wong, Keng Peng Tee, and Marcelo H. Ang. RMP-YOLO: A robust motion predictor for partially observable scenarios even if you only look once. In *2025 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 1024–1031, 2025. doi: 10.1109/ICRA55743.2025.11127743. URL https://doi.org/10.1109/ICRA55743.2025.11127743.

Jiawei Sun, Xibin Yue, Jiahui Li, Tianle Shen, Chengran Yuan, Shuo Sun, Sheng Guo, Quanyun Zhou, and Marcelo H. Ang. IMPACT: Behavioral intention-aware multimodal trajectory prediction with adaptive context trimming. *IEEE Robotics and Automation Letters*, 11(1):610–617, 2026. doi: 10.1109/LRA.2025.3634880. URL https://doi.org/10.1109/LRA.2025.3634880.

Balakrishnan Varadarajan, Ahmed Hefny, Avikalp Srivastava, Khaled S. Refaat, Nigamaa Nayakanti, Andre Cornman, Kan Chen, Bertrand Douillard, Chi Pang Lam, Dragomir Anguelov, and Benjamin Sapp. MultiPath++: Efficient information fusion and trajectory aggregation for behavior prediction. *arXiv preprint arXiv:2111.14973*, 2021. URL https://arxiv.org/abs/2111.14973.

Guangya Wan, Mingyang Ling, Xiaoqi Ren, Rujun Han, Sheng Li, and Zizhao Zhang. COMPASS: Enhancing agent long-horizon reasoning with evolving context. In Maria Liakata, Viviane P. Moreira, Jiajun Zhang, and David Jurgens (eds.), *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 3360–3380, San Diego, California, United States, July 2026. Association for Computational Linguistics. ISBN 979-8-89176-390-6. doi: 10.18653/v1/2026.acl-long.152. URL https://aclanthology.org/2026.acl-long.152/.

Guanzhi Wang, Yuqi Xie, Yunfan Jiang, Ajay Mandlekar, Chaowei Xiao, Yuke Zhu, Linxi Fan, and Anima Anandkumar. Voyager: An open-ended embodied agent with large language models. *arXiv preprint arXiv:2305.16291*, 2023. URL https://arxiv.org/abs/2305.16291.

Jiayu Wang, Weijiang Lv, Bowen Fu, Jing Fu, Jiayi Song, Lingyu Zhang, Lanxuan Xue, Luodi Chen, Zepeng Xin, Kaiyu Li, and Xiangyong Cao. Act as a real researcher: A suite of benchmarks evaluating frontier LLMs and agentic harnesses in research lifecycle, 2026. URL https://arxiv.org/abs/2606.07462.

Waymo Open Dataset. 2024 motion prediction challenge. Online, 2024. URL https://waymo.com/open/challenges/2024/motion-prediction/. Accessed: 2026-09-22.

Fan Wu, Cheng Chen, Zhenshan Tan, Taiyu Zhang, Xinzhen Xu, Yanyu Qian, Dingcheng Gao, Lanyun Zhu, Qi Zhu, Yi Tan, Deyi Ji, Guosheng Lin, Tianrun Chen, Deheng Ye, and Fayao Liu. Claw AI Lab: An autonomous multi-agent research team, 2026a. URL https://arxiv.org/abs/2605.22662.

Hongtao Wu, Boyun Zheng, Dingjie Song, Yu Jiang, Jianfeng Gao, Lei Xing, Lichao Sun, and Yixuan Yuan. Towards a medical AI scientist, 2026b. URL https://arxiv.org/abs/2603.28589.

Wenli Xiao, Jia Xie, Tonghe Zhang, Haotian Lin, Letian "Max" Fu, Haoru Xue, Jalen Lu, Yi Yang, Cunxi Dai, Zi Wang, Jimmy Wu, Guanzhi Wang, S. Shankar Sastry, Ken Goldberg, Linxi "Jim" Fan, Yuke Zhu, and Guanya Shi. ENPIRE: Agentic robot policy self-improvement in the real world, 2026. URL https://arxiv.org/abs/2606.19980.

Renjun Xu and Yang Yan. Agent skills for large language models: Architecture, acquisition, security, and the path forward, 2026. URL https://arxiv.org/abs/2602.12430.

Wanghan Xu, Shuo Li, Tianlin Ye, Qinglong Cao, Yixin Chen, Hengjian Gao, Yiheng Wang, Qi Li, Kun Li, Sheng Xu, Shengdu Chai, Fangchen Yu, Xiangyu Zhao, Zhangrui Zhao, Weijie Ma, et al. ResearchClawBench: A benchmark for end-to-end autonomous scientific research, 2026. URL https://arxiv.org/abs/2606.07591.

Yutaro Yamada, Robert Tjarko Lange, Cong Lu, Shengran Hu, Chris Lu, Jakob Foerster, Jeff Clune, and David Ha. The AI Scientist-v2: Workshop-level automated scientific discovery via agentic tree search. *arXiv preprint arXiv:2504.08066*, 2025. URL https://arxiv.org/abs/2504.08066.

Qi Yan, Brian Zhang, Yutong Zhang, Daniel Yang, Joshua White, Di Chen, Jiachao Liu, Langechuan Liu, Binnan Zhuang, Shaoshuai Shi, and Renjie Liao. TrajFlow: Multi-modal motion prediction via flow matching. In *2025 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 3923–3928, 2025. doi: 10.1109/IROS60139.2025.11247503. URL https://doi.org/10.1109/IROS60139.2025.11247503.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik R Narasimhan, and Yuan Cao. ReAct: Synergizing reasoning and acting in language models. In *The Eleventh International Conference on Learning Representations*, 2023. URL https://openreview.net/forum?id=WE_vluYUL-X.

Zikang Zhou, Hengjian Zhou, Haibo Hu, Zihao Wen, Jianping Wang, Yung-Hui Li, and Yu-Kai Huang. ModeSeq: Taming sparse multimodal motion prediction with sequential mode modeling. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 1612–1621, June 2025. URL https://openaccess.thecvf.com/content/CVPR2025/html/Zhou_ModeSeq_Taming_Sparse_Multimodal_Motion_Prediction_with_Sequential_Mode_Modeling_CVPR_2025_paper.html.

Qiran Zou, Hou Hei Lam, Wenhao Zhao, Tingting Chen, Yiming Tang, Samson Yu, Yingtao Zhu, Srinivas Anumasa, Zufeng Zhang, Tianyi Zhang, Chang Liu, Zhengyao Jiang, Anirudh Goyal, and Dianbo Liu. FML-bench: A controlled study of AI research agent strategies from the perspective of search dynamics, 2026. URL https://arxiv.org/abs/2605.17373.

A Metrics and Annotation Protocol

Benchmark mechanics and metric structure. While Section 4 distinguishes development from protected Soft mAP, both use the same WOMD metric implementation. Its empirical behavior is governed by two structural properties of the motion-forecasting evaluation engine:

1. **The 63-Cell Evaluation Decomposition:** Overall Soft mAP is computed as the unweighted arithmetic mean across 63 distinct evaluation cells: 3 agent classes (Vehicle, Pedestrian, Cyclist) $\times$ 3 evaluation time horizons ($3.0\,\text{s}, 5.0\,\text{s}, 8.0\,\text{s}$) $\times$ 7 scored trajectory-shape buckets (`STATIONARY`, `STRAIGHT`, `STRAIGHT_LEFT`, `STRAIGHT_RIGHT`, `LEFT_TURN`, `RIGHT_TURN`, `LEFT_U_TURN`). WOMD defines eight trajectory types, but its official metric merges `RIGHT_U_TURN` into `RIGHT_TURN` before precision–recall accumulation; therefore, only the seven buckets listed above are scored. Each cell contributes exactly $1/63 \approx 0.01587$ to the aggregate score. Because the evaluation handles agent classes independently, changing a parameter used exclusively by class

j cannot alter the score for class $i \neq j$. The class-level scores are therefore additive: Overall $= (\text{Veh} + \text{Ped} + \text{Cyc})/3$.

2. **Global Cross-Scene Sample Sorting:** Prediction confidences determine the ordering used to construct precision–recall curves across scenes. Additional matching predictions for a ground-truth trajectory are ignored by Soft mAP, whereas non-matching predictions remain false positives. Confidence calibration can therefore affect precision by changing the rank of non-matching predictions; any effect of per-agent normalization is an empirical property of the predictions, not a penalty for duplicate matches.

Conservative repeatability threshold and development validity. To avoid promoting changes that may reflect ordinary run-to-run variation, we prespecify a conservative development threshold $\tau \equiv \tau_{\text{promote}} = 0.005$ Soft mAP. We treat smaller changes as below the practical repeatability resolution of the protocol rather than as established improvements; this fixed threshold is applied uniformly across methods, model families, and research episodes.

A candidate is promoted only if it improves over its specific parent checkpoint on the 80-shard development split by ΔSoft mAP $\geq \tau$ ($+0.005$).

Attempt outcome taxonomy and UAR formulation. Every dispatched hypothesis $e \in \mathcal{E}$ executed by a sandboxed Coder receives exactly one terminal outcome label in the persistent archive $\mathcal{A}$:

- **Failed ($I_{\text{new}}(e) = 0$):** No valid development evaluation is produced within the repair budget. Causes include compilation errors, JAX/accelerator shape mismatches, out-of-memory failures, exhaustion of the three-cycle repair cap, and sandbox interceptions. The sandbox also aborts attempts to access the protected 70-shard split, modify evaluation code or assertions, submit an otherwise valid executable change that duplicates a previous experiment, import unauthorized external weights or datasets, or invoke unapproved system-level tools; the workspace is then reset to the parent state.
- **Non-improving ($I_{\text{new}}(e) = 1$):** A valid, non-duplicate experiment completes development evaluation but does not meet the promotion criteria: ΔSoft mAP $< \tau$. These results remain useful negative evidence and are consolidated into research memory $\mathcal{M}$.
- **Improving ($I_{\text{new}}(e) = 1$):** A valid, non-duplicate experiment achieves development $\Delta\text{Soft mAP} \geq \tau$.

The Unproductive Attempt Rate (UAR) is defined strictly by informational outcome:

$$\text{UAR} = \frac{\sum_{e \in \mathcal{E}} \mathbb{I}[e = \text{Failed}]}{|\mathcal{E}|} = 1 - \frac{\sum_{e \in \mathcal{E}} I_{\text{new}}(e)}{|\mathcal{E}|}, \tag{2}$$

Valid negative results (Non-improving) contribute informational value to memory $\mathcal{M}$ and are never counted against the system in UAR.

LLM tokens per valid experiment. We sum the provider-reported uncached input tokens and generated output tokens from all agent calls during a research episode, then divide by the number of attempts that produce a valid development evaluation. Provider-reported cache-read or cache-hit tokens are excluded, as are tokens used by the post-hoc Novelty judges.

Active Utilization Rate. We define the Active Utilization Rate as accelerator device-time spent executing training or model-inference workloads divided by the total accelerator device-time reserved over the research window. Compilation and tracing time do not count as active. Neither does any brief interval in which an accelerator remains reserved or on hold without executing training or inference, including the handoff between compilation and execution. Thus, this metric measures actual accelerator computation rather than reservation or occupancy.

Scope and Novelty. Every dispatched attempt produces at least an attempted patch before receiving a terminal outcome and is scored on two separate axes. Failed attempts are scored on their final attempted patch, so ambition is measured independently of execution success (Table 4). *Scope* measures what the diff touches and is computed deterministically, without an LLM. *Novelty* measures

Table 4: Rubric for attempted research changes. Scope is computed from the diff (see *Computing Scope*); Novelty is judged from the diff alone (Listing 1). A promoted change counts as a *verified mechanism-level discovery* if Scope ≥ 2, Novelty ≥ 2, and two blind human reviewers both confirm the Novelty level and the named closest precedent.

Level	**Scope** (what the diff touches; computed)	**Novelty** (relation to known practice; judged)
0	**Config.** After literal normalization the diff is empty: only numeric constants, flags, schedules, or configuration values change.	**Retune.** No new mechanism; an existing mechanism is re-parameterized or refactored with equivalent behavior.
1	**Local.** Substantive logic changes confined to a single function or class.	**Standard.** A textbook or library technique applied essentially as-is (e.g., label smoothing, Huber loss, Soft-NMS, EMA).
2	**Component.** Substantive changes across several functions, or a new function, class, or module, within one pipeline stage.	**Adapted.** Known technique(s) substantively restructured for the structure of motion forecasting (multimodal outputs, evaluation horizons, kinematics, agent classes, action tokenization), or composed to target a diagnosed failure.
3	**Cross-stage.** Substantive changes in two or more pipeline stages.	**Original.** No close precedent found after a retrieval-backed search; requires human confirmation.

how the introduced mechanism relates to known practice and is assigned by an LLM judge panel from the diff alone. The two axes are defined without reference to each other. They are correlated in practice, because retuning rarely requires broad edits, but a broad refactor can be Novelty 0 and a one-function change can be Novelty 2. Neither quantity is visible to the research loop; both are distinct from the in-loop exploration term $u(h, \mathcal{A})$ of Eq. 1.

Computing Scope. Every function and class in both repositories is labeled once, as part of the human-built infrastructure (Section 3.4), with one pipeline stage: preprocessing and features, action space and tokenization (MotionLM only), model architecture, training objective and optimization, or sampling and post-processing. Labels are assigned per symbol rather than per file, because some files mix stages; for example, loss overrides in `train_config.py` belong to the training-objective stage, while the literal assignments in the same file are configuration. Formatting and comments are removed, and string and numeric literals are replaced by typed placeholders. If the canonical diff is empty, Scope is 0. Otherwise, a line is *substantive* unless it is an import, a registration or wiring call, logging, or test code. Scope is then determined by the number of symbols and stages that contain substantive changes, following Table 4.

Judging Novelty. Each attempt is scored by both Gemini 3.6 Flash and Claude Opus 5; the reported score is the mean of the two, so every row of Table 1 is judged by the same panel. Each judge receives only the comment-stripped diff and the stage labels. Hypothesis records, agent notes, commit messages, and skill references are removed, because their form differs by harness and would reveal it. For every level ≥ 1, each judge must name the closest precedent and the difference from it. A provisional level 3 triggers a second, retrieval-enabled pass with a logged search.

Reported quantity. We report *Novelty*, the mean Novelty over all dispatched attempts, as a measure of research ambition. Scope is retained for the Single-Agent diagnostic analysis and case-study grading rather than aggregated across the primary comparisons. Two domain experts, blind to the originating harness, review every promoted change with Scope ≥ 2 and Novelty ≥ 2. On a stratified sample of 30 borderline candidates, their agreement on the Novelty level was Cohen's $\kappa = 0.88$.

Listing 1: System prompt for the Novelty judge panel. Scope is computed deterministically and is not judged.

```
You are an impartial reviewer of code changes to a motion-forecasting codebase
(Waymo Open Motion Dataset). You judge ONE axis: NOVELTY, i.e., how the mechanism
introduced by the diff relates to known practice. Do NOT judge how large the change
is, whether it is correct, or whether it is likely to help.

=== INPUT ===
[PIPELINE STAGES TOUCHED] {stage_labels}     # computed from the diff
[CODE DIFF]               {canonical_diff}   # comments, docstrings, agent notes removed

=== NOVELTY LEVELS ===
0 RETUNE    No new mechanism. Only values of existing parameters change, or existing
            logic is refactored with equivalent behavior.
1 STANDARD  A textbook or library technique inserted essentially as-is
            (e.g., label smoothing, Huber loss, Soft-NMS, cosine schedule, EMA).
2 ADAPTED   One or more known techniques substantively restructured for motion
            forecasting (multimodal outputs, evaluation horizons, kinematics,
            agent classes, action tokenization), or composed to address a
            specific failure mode.
3 ORIGINAL  You cannot identify a close precedent after searching. Use sparingly.

=== PROCEDURE ===
1. Describe what the diff actually does in two sentences, from the code alone.
2. Name the closest precedent (technique and, if known, paper) for the mechanism.
   For level 3, list the searches you ran and why the results are not close.
3. State how the diff differs from that precedent; assign the level that this
   difference supports.
4. Judge the code, not the identifiers: a class with an ambitious name that wraps
   a library call is level 1 at most.

=== OUTPUT (JSON only) ===
{
  "diff_summary": "<two sentences>",
  "closest_precedent": "<technique / paper, or null for level 0>",
  "delta_from_precedent": "<what the diff changes relative to it>",
  "search_log": ["<query>", "..."],
  "novelty_score": <0|1|2|3>,
  "confidence": "<low|medium|high>"
}
```

Listing 2: Declarative JSON schemas for domain observability tools.

```
[
{
  "name": "render_scenario_bev",
  "description": "Renders a static 2D Bird's Eye View (BEV) PNG overlay showing the HD map,
     past history, GT future, and 6 predicted trajectory modes with probabilities.",
  "parameters": {
    "scenario_id": {"type": "string", "description": "Unique WOMD scenario ID"},
    "output_path": {"type": "string", "description": "File path destination for .png"},
    "side_by_side": {"type": "boolean", "default": false, "description": "Render side-by-side
     baseline vs candidate comparison"}
  },
  "returns": {
    "rendered": "boolean",
    "top1_displacement_error_m": "float",
    "mode_spread_variance": "float",
    "dominant_failure_mode": "string"
  }
},
{
  "name": "slice_errors",
  "description": "Extracts 63-cell AP decomposition and cross-scene failure slices.",
  "parameters": {
    "run_id": {"type": "string", "description": "Experiment run or checkpoint identifier"},
    "slice_axis": {"type": "string", "enum": ["agent_class", "horizon", "trajectory_bucket", "
     speed_regime"]}
  },
  "returns": {
    "ap_matrix": {"type": "object", "description": "Detailed AP matrix for the requested slice
     "},
    "worst_cell": {"type": "string", "description": "Evaluation cell with lowest AP relative
     to baseline"},
    "false_positive_rate": {"type": "float"}}}]
```

Table 5: Role-scoped capabilities. ✓: full access; R: read-only; S: supplied as a bounded excerpt ($\leq$32K tokens) attached by the Evolution Manager; –: no access. The Aggregator is deterministic code within the Researcher team, not an LLM agent. The Harness is non-agent infrastructure. *Baselines* gives the envelope each baseline scaffold receives as a whole; only DRIVELAB partitions it across roles. All restrictions are enforced by the mechanism in the last column, not by prompting.

Capability	Researcher	Coder	Evol. Mgr.	Harness	Baselines	Enforced by
Code and workspace						
Read / search repository	R	✓[a]	–	–	✓	file permissions
Edit model, training, post-processing code	–	✓[a]	–	–	✓	sandbox checkout
Edit evaluation code, metrics, or assertions	–	–	–	–	–	read-only mount; sandbox abort
Access other Coders' workspaces	–	–	–	–	n/a	sandbox isolation
Import external weights or datasets	–	–	–	–	–	network policy; sandbox abort
Execution and evaluation						
Config checks, unit tests, 10-shard smoke tests	–	✓	–	–	✓	job API scope
Launch / monitor training jobs	–	✓[b]	–	–	✓	job API scope
Development evaluation (80 shards)	–	✓	–	–	✓	job API scope
Metrics and diagnostics of completed runs	R	✓	R	–	✓	job API scope
Observability tools (BEV, slices, failure mining)	✓	✓	–	–	✓	read-only tool API
Protected evaluation (70 shards)	–	–	–	✓[c]	–	not mounted; no credentials
Research state						
Propose and review hypotheses	✓	–	–	–	✓	role tool set
Rank and dispatch pairs to Coders (Eq. 1)	✓[d]	–	–	–	✓	role tool set
Read archive	✓[e]	S	✓	–	–[f]	state-store API
Read research memory	S	–	✓	–	–[f]	state-store API
Write archive and memory	–	–	✓	–	–	state-store API
Maintain parent pool; apply promotion rule (τ)	R	–	✓	–	✓[g]	state-store API
Remaining budget	R	–	✓	–	✓	state-store API
Knowledge and external tools						
Read skill library	R[h]	✓	–	–	✓[i]	file permissions
Write skill library (distillation)	–	✓[j]	–	–	–	file permissions
External web search	✓	–	–	–	✓	role tool set

[a] Own isolated checkout of the parent state only. [b] Coders launch jobs on accelerators; the Aggregator dispatches hypotheses to Coders, not jobs. [c] After search ends: once on the final development-selected incumbent, plus the reported post-hoc case-study component evaluations; scores are never returned to any agent. [d] Performed by the rule-based Aggregator. [e] Proposers query the archive for related attempts. [f] Baselines keep history only in their working context, subject to compaction. [g] Same τ rule, applied within each baseline scaffold. [h] Diagnostic workflows only (e.g., `womd-failure-diagnosis`), invoked through the observability API. [i] Expert seed skills only; no in-episode distillation. [j] Distillation runs after any Coder subtask exceeding 10 steps.

B INFRASTRUCTURE AND TOOL INTERFACES

Compute and cluster execution architecture. All agent-driven experimentation executes on a fixed accelerator allocation through a managed JAX/Flax job interface. Training data, checkpoints, and job execution are exposed through the same shared infrastructure to all compared methods.

Role permissions. To prevent prompt contamination and structural cheating, role boundaries are enforced strictly at the API and file-permission layers (Table 5).

Domain observability schemas. The harness exposes read-only diagnostic tools for scene visualization, sliced error analysis, failure-case retrieval, training-curve inspection, and parent–candidate comparison. Listing 2 specifies the input/output schemas for the primary visual and statistical tools.

C DRIVELAB HARNESS SUMMARY

Algorithm 1 summarizes one research episode, Algorithm 2 the Coder procedure, and Table 6 the harness parameters stated throughout the paper. Hypotheses go directly from the Researcher team to the Coders, and results come directly back; the Evolution Manager observes the loop asynchronously, maintains the archive, research memory, pool of eligible parents, and budget, and supplies bounded context to the other roles.

Algorithm 1 One DRIVELAB research episode.

Require: Seed state s_0, research brief, compute budget, $k = 4$ parallel Coders, $\tau = 0.005$, $\beta_t = 1.5 \cdot 0.92^t$

Initialize the archive $\mathcal{A}$ and research memory $\mathcal{M}$
Set eligible parents to $\{s_0\}$, skill library to expert seed, and $t \leftarrow 0$
while budget remains **and** stopping criterion is not met **do**
    Proposers generate hypotheses from brief, $\mathcal{M}$, parents, diagnostics, budget, archive, and web
    Reviewers score each hypothesis; $q(h \mid s_p) \leftarrow$ mean review score
    Aggregator ranks parent–hypothesis pairs by score_t (Eq. 1)
    Dispatch top-k pairs fitting budget to Coders (Algorithm 2)
    Manager attaches relevant prior attempts to each dispatched pair
    $t \leftarrow t + 1$
end while

Asynchronously (whenever a Coder returns):
Researcher team receives the Coder's report
Manager records the full returned state in $\mathcal{A}$, updates eligible parents and budget ▷ Promotion requires $\Delta f_{\mathrm{dev}} \geq \tau$ over parent
After every four completed experiments, **Manager** consolidates $\mathcal{M}$

Algorithm 2 Coder procedure for one dispatched pair (s_p, h).

procedure CODER(s_p, h)
    Checkout isolated s_p with h, attached prior attempts, and relevant skills
    Implement h
    for $c = 0$ **to** 3 **do** ▷ Initial attempt plus up to three repair cycles
        if $c > 0$ **then**
            Diagnose failure from available artifacts and repair implementation
        end if
        Run inexpensive checks (config validation, unit tests, small-shard smoke tests)
        if all checks pass **then**
            Launch training (Appendix D) and development evaluation
            Monitor logs, checkpoints, metrics, and job health
            if code and config ran as intended and produced expected artifacts/metrics **then**
                return validated diff, configuration, metrics, diagnostics, and report
            end if
        end if
    end for
    return failure stage and attempted repairs
end procedure

Throughout: After any subtask taking > 10 steps, distill reusable knowledge into skill library.

D EXPERIMENTAL DETAILS

Baseline Implementations. Single-Agent AutoResearch (Karpathy). We adapt the Single-Agent AutoResearch recipe introduced by Karpathy (Karpathy, 2026), in which one coding agent repeatedly inspects the current implementation and experimental results, proposes a change, modifies the code, launches training, evaluates the resulting model, and decides how to proceed. The original setup is designed for lightweight language-model experiments and allocates a fixed **5-minute training budget** to each candidate. Motion-prediction training is substantially more expensive, so we replace that fixed limit with the matched training-mode policy below: warm-started candidates train for at most two hours, checkpoint-incompatible candidates use the full seed recipe, and post-processing-only candidates require no retraining. This preserves the original edit–train–evaluate loop while providing a sufficiently stable evaluation signal for comparing changes to Wayformer and MotionLM. All baseline methods follow the same training-mode, cost-accounting, and evaluation rules. This is the baseline labeled **Single-Agent** in the main experiments.

Table 6: DRIVELAB harness parameters, collected from the main text and appendices.

Parameter	Value	Defined in
Top-k pairs; parallel Coders	$k = 4$	Sec. 3.1; App. D
Exploration schedule	$\beta_t = \beta_0 \gamma^t$, $\beta_0 = 1.5$, $\gamma = 0.92$	Eq. 1; App. D
Promotion threshold	$\tau = 0.005$ dev Soft mAP	Sec. 4.1; App. A
Repair cycles per hypothesis	at most 3	Sec. 3.2
Training budget per experiment	Warm start $\leq$ 2 h; from-scratch $\approx$ 3 h (MotionLM) / 10 h (Wayformer)	App. D
Smoke tests	10-shard dev subset; 100-step run	App. D; App. F
Skill-distillation trigger	subtask longer than 10 steps	Sec. 3.2
Memory consolidation	every 4 completed experiments	App. D
Memory and archive context cap	32K tokens per role	Sec. 3.3
Development / protected shards	80 / 70	Sec. 4.1
Fixed-time budget	24 hours on 4 accelerators	Sec. 4.1
Open-ended stopping rule	10 consecutive valid non-improving evaluations or 12 hours without improvement	Sec. 4.1

Antigravity scaffold. As a separate baseline, we use **Google Antigravity** (Google, 2026) and invoke its `/goal` workflow with the research objective and repository instructions. The `/goal` workflow instructs the agent to continue autonomously until the specified objective is completed, including editing code, running commands, diagnosing failures, and applying subsequent corrections without requiring turn-by-turn human input. This is the baseline labeled **Antigravity Scaffold** in the main experiments; it is distinct from the Karpathy AutoResearch baseline described above.

Role-based multi-agent system. We additionally construct a role-based multi-agent baseline following the general design philosophy of multi-agent frameworks such as OWL (Hu et al., 2025) and MetaGPT (Hong et al., 2024), which decompose complex tasks across specialized agents rather than assigning the entire workflow to a single agent. We adapt this idea to autonomous model research using three roles: a **Planner**, a **Coder**, and a **Reviewer**. The Planner examines the current model, experimental history, and evaluation results and proposes the next research hypothesis and implementation plan, organized as a search tree. The Coder is responsible for carrying out the plan, including modifying the repository, debugging the implementation, and launching training and evaluation. The Reviewer independently checks the resulting implementation and experimental outputs, identifies failures or inconsistencies, and returns feedback before the team proceeds to the next experiment. The agents communicate through explicit role-based handoffs, providing planning, execution, and verification within each research iteration.

This baseline captures the principal benefit of conventional role specialization while remaining distinct from DRIVELAB. In particular, it does not include the persistent Evolution Manager, archive-based candidate selection, exploration–exploitation management, or cross-experiment skill distillation introduced by our method. Consequently, it tests whether simply replacing a single research agent with a structured team of specialized agents is sufficient for sustained long-horizon model improvement.

Matched capabilities and parallelism. All baselines receive the same agent-visible repository interfaces, diagnostic and observability tools, and initial library of reusable implementation skills provided to DRIVELAB. This matches access to seed skills; the baselines do not receive skills distilled by DRIVELAB during an episode. Episode-level skill distillation is a component of DRIVELAB evaluated separately in the ablations. To avoid confounding the comparison with hardware parallelism, we also explicitly prompt every baseline to launch multiple independent training jobs whenever accelerator capacity is available, subject to the same four-accelerator allocation.

Datasets and splits. Experiments use the public Waymo Open Motion Dataset (WOMD) v1.2.0 (Ettinger et al., 2021). All seed checkpoints and trainable candidates use the full official training split. We deterministically partition the official 150-shard validation set into 80 development shards and 70 protected shards before any research episode begins. The 80-shard development partition is the sole source of model-quality feedback during research: candidate evaluation, promotion, parent selection, sliced diagnostics, and plateau detection all use this partition. A deterministic 10-shard subset of these 80 shards is used for smoke tests and fast diagnostics, but not as the

sole basis for final candidate promotion. The remaining 70 shards are excluded from agent-visible storage mounts and evaluation credentials. After an episode terminates, the harness evaluates that episode's single final development-selected incumbent once on the protected split and performs the reported post-hoc evaluations of the case-study components. Protected results are never returned to agents and never used to resume or redirect search. Thus, all agent-visible model-quality feedback is computed on the fixed 80-shard development split; unless explicitly labeled as a development result, reported final model-quality scores are post-episode evaluations on the fixed 70-shard protected split. The 150 shards denote only the union of these two partitions and are not used as a separate held-out evaluation set.

Official WOMD Test-Set Result. Main-text model-quality scores reported for final checkpoints are protected-split scores. Figure 3 is the exception: its curves show development-split scores observed throughout search. The case-study component scores in Section 4.4 are post-hoc protected evaluations, and the candidate comparisons in Appendix G are development-split results. As an external check, we evaluated a single DRIVELAB-improved model on the official WOMD motion-prediction test set. Operating without ensembling or LiDAR inputs, our model achieves 0.4523 Soft mAP. To preserve double-blind anonymity, we have not made this submission or score public, associated it with a publicly visible team or method entry, or included identifying test-server metadata.

Among published single-model results under the same no-ensemble/no-LiDAR restriction, this outperforms ModeSeq (0.4487) (Zhou et al., 2025) and EDA (0.4510) (Lin et al., 2024). It remains competitive with, though slightly below, recent architectures such as UniMotion (0.4642) (Song et al., 2025), RMP-YOLO (0.4673) (Sun et al., 2025), TrajFlow (0.4710) (Yan et al., 2025), and IMPACT (0.4721) (Sun et al., 2026). It also naturally trails the best reported ensemble methods (0.4737–0.4801) (Zhou et al., 2025; Sun et al., 2025; 2026).

Notably, these baselines are purpose-built, human-designed architectures, whereas our model was obtained entirely via automated research from a low-performing seed checkpoint. While differences in training setups, inputs, and evaluation-server versions make this comparison indicative rather than exact, the results demonstrate that DRIVELAB can autonomously evolve highly competitive models.

Model architectures and seed checkpoints.

- **Wayformer Baseline:** Attention-based scene encoder with query-based non-autoregressive trajectory decoder (Nayakanti et al., 2022). Seed checkpoint trained to 500k steps on accelerator pods ($\mathrm{Soft\,mAP} = 0.3788$, $\mathrm{minADE} = 0.605$ m).
- **MotionLM Baseline:** Autoregressive discrete sequence model generating multi-agent motion tokens over a $13 \times 13 = 169$ Cartesian grid (Seff et al., 2023). The seed checkpoint used by all runs in Table 1 is the standard baseline (700k steps, standard cross-entropy, $\mathrm{Soft\,mAP} = 0.3905$, $\mathrm{minADE} = 0.571$ m).

Training modes. Each candidate inherits its parent's code and configuration. By default, it warm-starts from the parent's checkpoint and trains within the 2-hour per-experiment budget. When a change makes the parent checkpoint unusable (e.g., reshaped parameters or a changed action vocabulary), the candidate is instead retrained from scratch with the seed recipe, which takes about 3 hours for MotionLM and 10 hours for Wayformer; this cost is included in the hypothesis's cost estimate and charged to the same 24-hour episode budget. Post-processing-only candidates are re-evaluated on the parent's checkpoint without retraining. Baselines follow the same rules. In accordance with Section 3.4, all models are evaluated as single models rather than multi-model ensembles to dedicate the research budget to hypothesis exploration.

Hyperparameters and search policies. The exploration parameter β_t in Eq. 1 is initialized at $\beta_0 = 1.5$ and annealed via $\beta_t = \beta_0 \cdot (0.92)^t$. Sandboxed Coder concurrency is fixed at $k = 4$ parallel workers. The Coder repair cap is set to 3 cycles; failures exceeding 3 iterations are terminated and recorded as `Failed`. Memory consolidation occurs after every completed batch of 4 experiments.

Rule-based hypothesis similarity. For the exploration term in Eq. 1, each candidate is represented by a canonical edit signature containing its target pipeline stage, referenced files and functions, and tokenized implementation sketch. An archived attempt uses the same fields together with the

changed lines from its committed diff, or its attempted patch and failure category when execution fails. Formatting and comments are removed, identifiers are split into lexical tokens, and string and numeric literals are replaced by typed placeholders so scalar retuning of the same mechanism remains similar. With $T(\cdot)$ denoting the resulting token sequence, similarity is

$$\mathrm{sim}(h, e) = 1 - \frac{D_{\mathrm{lev}}(T(h), T(e))}{\max(|T(h)|, |T(e)|, 1)}, \tag{3}$$

where D_{lev} is unit-cost token-level Levenshtein distance. This deterministic construction lies in $[0, 1]$ and uses neither a learned encoder nor an LLM judge.

Additional Motion-Prediction Metrics. Table 7 supplements Table 1 with the five additional metrics for the same settings and checkpoints averaged.

Table 7: Additional prediction metrics. Soft mAP values are from Table 1.

Base LLM	Method	minADE (m) ↓	minFDE (m) ↓	Miss (%) ↓	mAP ↑	Soft mAP ↑	Overlap (%) ↓
Wayformer							
—	Initial seed	0.605	1.276	14.6	0.366	0.3788	13.7
Gemini 3.6 Flash	Single-Agent	0.601	1.245	14.3	0.374	0.3855	13.6
	Role-based MAS	0.598	1.215	14.0	0.385	0.3951	13.6
	Antigravity Scaffold	0.597	1.212	13.7	0.391	0.3998	13.5
	DRIVELAB (ours)	0.592	1.202	13.3	0.405	0.4128	13.4
Claude Opus 5	Single-Agent	0.600	1.232	14.1	0.377	0.3875	13.6
	DRIVELAB (ours)	0.589	1.196	13.1	0.411	0.4188	13.3
Gemini 3.1 Pro	Single-Agent	0.603	1.258	14.4	0.371	0.3830	13.7
	DRIVELAB (ours)	0.593	1.204	13.3	0.404	0.4125	13.4
MotionLM							
—	Initial seed	0.571	1.179	14.5	0.373	0.3905	13.0
Gemini 3.6 Flash	Single-Agent	0.564	1.163	13.4	0.380	0.4037	12.9
	Role-based MAS	0.562	1.158	13.0	0.388	0.4118	12.8
	Antigravity Scaffold	0.559	1.153	12.9	0.390	0.4146	12.8
	DRIVELAB (ours)	0.553	1.143	12.6	0.436	0.4432	12.7
Claude Opus 5	Single-Agent	0.556	1.148	12.8	0.400	0.4228	12.8
	DRIVELAB (ours)	0.552	1.140	12.5	0.445	0.4515	12.7
Gemini 3.1 Pro	Single-Agent	0.565	1.167	13.5	0.377	0.4015	12.9
	DRIVELAB (ours)	0.555	1.146	12.7	0.421	0.4296	12.7

E How Single-Agent Baselines Fail

Section 1 identified three recurring failure modes when long-horizon research is framed as a flat, unmanaged edit–train–evaluate loop: *context overload and instruction drift*, *procedural rediscovery*, and *search narrowing*. This appendix documents the pilot experiments behind that observation. Table 8 summarizes 24-hour Single-Agent runs on both codebases; the three diagnostic columns map onto the three failure modes. *Failed* is the percentage of all dispatched experiments that fail to produce a valid evaluation, including duplicate attempts as defined in Appendix A; repeated repository-specific runtime errors are a major cause (procedural rediscovery). *Non-improving* is the percentage of valid experiments that do not meet the development improvement criterion, which the ledgers most often trace to constraints or negative results dropped from the working context (instruction drift). Writing $N_{\mathrm{all}} = N_{\mathrm{failed}} + N_{\mathrm{valid}}$, the two percentages are $100N_{\mathrm{failed}}/N_{\mathrm{all}}$ and $100N_{\mathrm{nonimp}}/N_{\mathrm{valid}}$, respectively. They have different denominators and should not be added: the share of all attempts that are valid but non-improving is $(1 - f)n$, where f and n are the two reported rates expressed as fractions. *Scope-0/1 Share* is the fraction of attempts rated as trivial or hyperparameter-level changes on the Scope axis of the rubric in Table 4, and directly measures search narrowing. On both codebases, roughly three-quarters of attempts are low-scope, and only about one in seven attempts improves the metric. Specifically, the improving share of all attempts is $(1 - f)(1 - n)$, approximately 14.3% for Wayformer and 10.0% for MotionLM. Below we walk through two auditable ledger cases that illustrate the first two failure modes concretely.

Failure example A (procedural rediscovery): JAX export tensor shape mismatch. In one run, the agent introduced a trajectory-type-aware aggregation change that passed an isolated unit test but

Table 8: Pilot failure diagnostics for Single-Agent baselines across 24-hour runs (Gemini 3.6 Flash). Failed is measured over all dispatched experiments; Non-improving is measured only over valid experiments. These percentages have different denominators and do not sum to an outcome distribution.

Codebase	Net ΔSoft mAP	Failed (% of all)	Non-improving (% of valid)	Scope-0/1 Share (%)
Wayformer	+0.0063	21.4	81.8	78.6
MotionLM	+0.0118	25.0	86.7	75.0

failed during compiled export because an eight-entry vector over the full WOMD trajectory-type taxonomy was multiplied by a seven-entry vector over the scored trajectory-shape buckets obtained after the official `RIGHT_U_TURN`-to-`RIGHT_TURN` merge.

Listing 3: Sanitized single-loop execution failure: array broadcast mismatch during compiled export.

```
Exporting candidate model...
Traceback (most recent call last):
  File "<aggregation_module>", in _run_single_agent
TypeError: mul got incompatible shapes for broadcasting: (8,), (7,)
```

The repository convention therefore requires operations over evaluation buckets to merge and align the full eight-type taxonomy into the seven scored buckets before elementwise computation. The agent had no way to retain this convention across attempts. It spent its three-cycle repair budget modifying launch parameters rather than diagnosing the mismatch, consuming hours with zero experimental yield; a later attempt in the same run hit a variant of the same error. In DRIVELAB, the failure signature and fix were recorded in the skill library after the first occurrence and did not recur.

Failure example B (context overload and instruction drift): silent agent-type plumbing defect. In a separate run, the agent attempted class-specific suppression in the post-processor. Its implementation invoked the aggregator without passing the feature dictionary containing agent types, and the underlying module silently defaulted missing types to the vehicle class (Listing 4).

Listing 4: Sanitized single-loop plumbing defect: agent types never reach the aggregator, so the module falls back to its vehicle default.

```
# post_processor.py
suppressed = jax.vmap(self.aggregator)(sampled_traj)   # features not passed

# aggregation_module.py (existing)
def __call__(self, traj, features=None):
    agent_types = (features["agent_type"] if features is not None
                   else jnp.ones(traj.shape[0], dtype=jnp.int32))  # 1 == VEHICLE
    params = self.class_params[agent_types]
    ...
```

All pedestrians and cyclists were therefore processed with vehicle parameters for 14 consecutive experiments. Because the evaluation harness still produced valid numbers, the defect went unnoticed; the per-class metrics in the ledger (Listing 5) are identical across experiments that nominally varied pedestrian and cyclist parameters, yet the agent concluded that pedestrian Soft mAP was insensitive to suppression. The original hypothesis notes that would have exposed the discrepancy had already been flushed by context-window compaction, so the direction was abandoned on a false negative.

Listing 5: Sanitized ledger excerpt for the same run. Pedestrian and cyclist metrics do not change across experiments that nominally modified their suppression parameters.

```
exp_id   hypothesis                                ΔPed Soft mAP   ΔCyc Soft mAP
e01     ped suppression radius 2.0 -> 3.0          0.0000          0.0000
e02     ped suppression radius 3.0 -> 4.5          0.0000          0.0000
e03     cyc suppression weight 0.5 -> 0.8          0.0000          0.0000
...
e14     ped/cyc joint schedule                      0.0000          0.0000
[agent note, e14]: "Pedestrian Soft mAP appears insensitive to
suppression radius; deprioritizing this direction."
```

In DRIVELAB, the Evolution Manager's persistent archive retains hypothesis records and diagnostics outside the Coder's context, and the driving-specific observability tools surface per-class metrics that flag uniform behavior after the first run.

Failure example C (search narrowing): retreat to hyperparameter tuning after a failed pre-processor edit. In a third run, the agent began ambitiously. In round 3 it proposed replacing the fixed-radius agent-neighbor selection in the pre-processor with a distance- and heading-aware selection, a mechanism-level change touching the feature encoder and the batching pipeline. The implementation failed twice (Listing 6): first on a padding-mask shape error, then on an out-of-memory error when the variable neighbor count broke static batching.

Listing 6: Sanitized failures of the pre-processor edit (round 3). The second failure arises because variable neighbor counts break static batching.

```
# attempt 1
  File "<preprocessor_module>", in select_neighbors
ValueError: Incompatible shapes for broadcasting: mask (B, 64) vs
scores (B, 48, 2)

# attempt 2
  File "<batching_module>", in stack_features
XlaRuntimeError: RESOURCE_EXHAUSTED: Out of memory while trying to
allocate 21474836480 bytes (ragged neighbor set padded to max=512).
```

After the second failure the agent abandoned the idea, and its subsequent hypotheses were markedly narrower (Listing 7): the remaining eleven rounds consisted entirely of adjusting the learning-rate schedule, the neighbor count under the *existing* fixed-radius rule, and the trajectory-sampling temperature. Each ran successfully but moved $\mathrm{Soft\,mAP}$ by less than ± 0.002, and none was rated above Scope 1. Each proposal was evaluated against the current promoted checkpoint and reverted when its gain fell below $\tau_{\mathrm{promote}} = 0.005$.

Listing 7: Sanitized hypothesis log for the same run after the second failure. Every subsequent proposal is a Scope-0/1 adjustment to the current promoted checkpoint; sub-threshold edits are reverted.

```
round  status  hypothesis                                  scope  Delta_Soft_mAP
  3    FAIL    heading-aware neighbor selection (preproc)    3      --
  3    FAIL    retry with ragged padding                     3      --
  4    REVERT  lr peak 3e-4 -> 2e-4                          0   -0.0011
  5    REVERT  warmup 1k -> 2k steps                         0   +0.0004
  6    REVERT  fixed-radius num_neighbors 48 -> 64           1   +0.0009
  7    REVERT  sampling temperature 1.0 -> 0.9               1   -0.0006
  8    REVERT  lr peak 3e-4 -> 2.5e-4                        0   +0.0002
 ...
 14    REVERT  num_neighbors 48 -> 56                        1   -0.0003
[agent note, round 4]: "Architectural changes are too risky given the
remaining budget; focusing on stable configuration adjustments."
```

A single failed structural attempt thus converted the rest of a 24-hour run into local tuning, which is precisely the pattern the Scope-0/1 column in Table 8 aggregates. In DRIVELAB, this dynamic is counteracted on two fronts: the exploration term u in Equation 1 lowers the rank of hypotheses that repeat the edit pattern of earlier attempts, and the failed pre-processor patch is stored in the archive with its failure report, so it can be supplied to later Coders as a relevant prior attempt.

F AGENT SKILLS

Following the open Agent Skills specification (`SKILL.md`), each Coder skill is a self-contained directory containing a Markdown instruction file with YAML frontmatter followed by structured procedural sections, architectural boundary rules, reference templates, and operational guardrails. Coders retrieve relevant implementation skills before specialized implementation and debugging tasks. Read-only scenario diagnosis is exposed separately to the Researcher through the observability interface. Below we present two representative Coder skills and one Researcher diagnostic workflow, with their provenance made explicit: (1) the accelerator smoke-testing workflow is an expert-authored seed skill available at episode initialization; (2) the discrete action-space boundary

workflow is automatically distilled during an episode; and (3) the scenario-level WOMD failure-diagnosis workflow with Bird's-Eye-View (BEV) visualization is an expert-authored seed skill available at episode initialization.

1. Fast smoke-testing via interactive kernel hot-reloading (`interactive-kernel-smoke-test`; expert seed skill). Building a standalone training binary and scheduling a multi-node accelerator job incurs a 15–20-minute compilation overhead. To verify JAX graph tracing, tensor broadcasting, and gradient finiteness in under 90 seconds, the Coder invokes the `interactive-kernel-smoke-test` skill (Listing 8), which connects to a live accelerator kernel, hot-reloads modified workspace modules in-memory, and executes a 100-step verification run.

Listing 8: The `interactive-kernel-smoke-test/SKILL.md` specification used by the Coder to validate model and loss edits in-memory prior to full-scale training.

```
---
name: interactive-kernel-smoke-test
description: >-
  Connects to an interactive TPU/GPU kernel runtime and executes a 100-step
  training smoke test by hot-reloading modified local workspace modules in-memory.
  Use before launching full training runs to verify XLA compilation, tensor shapes,
  and finite loss gradients without rebuilding standalone binary packages.
---

# Interactive Kernel Hot-Reload Smoke Test Workflow

CRITICAL RULE: Interactive kernels are strictly for smoke testing (100-200 steps).
Do NOT run full 700k-step training in an interactive session; single-host data loaders
will exhaust host memory at epoch boundaries without distributed dataset workers.

## 1. Boundaries & Hot-Reloading Guardrails
- Task Registry De-duplication: Reloading dataset modules raises DuplicateProviderError
  unless Registry entries are explicitly removed before calling importlib.reload().
- ABI Boundary Protection: Restrict workspace_hot_reload strictly to pure-Python model
  packages (package_restrict="motion_forecasting"). Dynamically reloading C++ or XLA
  extensions causes ABI mismatches that crash the accelerator runtime.
- Asynchronous Execution: Wrap train_and_evaluate in a background threading.Thread(daemon=True)
  so XLA compilation does not trigger tool RPC timeouts; poll SUCCESS.txt or ERROR.txt.
- Trace vs. Runtime Debugging: Standard print() executes only once during JAX tracing.
  Use jax.debug.print("step_loss: {}", loss) to verify active accelerator execution.

## 2. Reference Execution Snippet (Python)
import importlib, linecache, os, threading, traceback
import jax

def run_smoke_test_in_background(workdir: str, smoke_steps: int = 100):
  try:
    with workspace_hot_reload(package_restrict="motion_forecasting"):
      linecache.clearcache()
      from motion_forecasting.modules import embedding, aggregation
      from motion_forecasting.datasets import task_registry
      from motion_forecasting.experiments import train_config

      # Guardrail: Clear registered tasks BEFORE reloading dataset modules
      for name in list(task_registry.names()):
        task_registry.remove(name)

      for mod in (embedding, aggregation, task_registry, train_config):
        importlib.reload(mod)

      cfg = train_config.build_autoregressive_experiment(
          train_steps=smoke_steps, eval_interval=smoke_steps, workdir=workdir
      )
      exp = cfg.instantiate()
      exp.trainer.train_and_evaluate(train_steps=smoke_steps, rng=jax.random.PRNGKey(0))
    open(os.path.join(workdir, "SUCCESS.txt"), "w").write("Smoke test passed.\n")
  except Exception:
    open(os.path.join(workdir, "ERROR.txt"), "w").write(traceback.format_exc())

threading.Thread(
    target=run_smoke_test_in_background, args=("/tmp/smoke_test", 100), daemon=True
).start()
```

2. Action-space boundary and tokenization debugging (`motionlm-action-space`; automatically distilled skill). In autoregressive motion forecasting, continuous trajectories are quantized into a 13×13 Cartesian Verlet-action grid with 169 discrete tokens. Each token represents a quantized second-order position increment (an acceleration-like action), and center token `#84` represents zero acceleration. Without explicit architectural guidance, coding agents frequently attempt to apply continuous spatial rotations inside the training loss function—after ground-truth trajectories have already been quantized into integer indices in $[0, 168]$—or alter grid geometry in ways that invalidate pre-tokenized datasets. Listing 9 shows the `motionlm-action-space/SKILL.md` specification that enforces the "Boundary vs. Internal Math" decision rule.

Listing 9: The `motionlm-action-space/SKILL.md` specification governing discrete Verlet vocabulary edits vs. configuration-level loss regularizers.

```
---
name: motionlm-action-space
description: >-
  Workflow and architectural boundary rules for implementing research ideas in
  the discrete action space (13x13 Verlet-action vocabulary, 169 tokens,
  and logit regularization). Use when modifying token loss functions in
  train_config.py or continuous-to-discrete quantization in discrete_action.py.
---

# Stage 2: Discrete Action Space & Vocabulary Workflow

## 1. Architectural Decision Rule ("Boundary vs. Internal Math")
Before editing any file, apply these 3 Acid Tests to select the correct implementation tier:
1. Scalar vocabulary sizes, learning schedules, or loss weights? -> train_config.py
2. Modifying input batch tensors OR regularizing output action_logits? -> train_config.py
   (Subclass AutoregressivePrediction and override compute_loss).
3. Altering continuous-to-discrete binning or 13x13 Verlet grid geometry? -> discrete_action.
    py

## 2. Reference Pattern: Logit Regularization in train_config.py
In the 13x13 grid (num_actions=169), token #84 is the central zero-change token (6, 6).
Low-speed vulnerable road users often collapse onto overconfident predictions of token #84.
Regularize logit probabilities in compute_loss without mutating pre-tokenized integer targets:

class ActionLabelSmoothingPrediction(CombinedAutoregressivePrediction):
  """Regularizes 169-token Verlet logits while preserving repulsive loss terms."""

  def compute_loss(self, batch, action_logits):
    valid_mask = batch['output/target_valid_mask']
    target_actions = batch['output/target_actions']
    num_classes = action_logits.shape[-1]  # 169 discrete Verlet tokens
    one_hot = jax.nn.one_hot(target_actions, num_classes)
    smooth_targets = (1.0 - 0.05) * one_hot + (0.05 / num_classes)
    log_probs = jax.nn.log_softmax(action_logits, axis=-1)
    step_loss = -jnp.sum(smooth_targets * log_probs, axis=-1)
    base_loss = jnp.sum(step_loss * valid_mask) / jnp.maximum(jnp.sum(valid_mask), 1.0)
    _, metrics = super().compute_loss(batch, action_logits)
    total_loss = base_loss + 0.10 * metrics['repulsive_loss']
    metrics['loss'] = total_loss
    return total_loss, metrics

## 3. Common Pitfalls & Debugging Gotchas
- Never Rotate target_actions in loss_fn: By the time tensors reach loss_fn,
  batch['output/target_actions'] are integer tokens in [0, 168]. Continuous coordinate
  rotations must be implemented upstream in agent_preprocessors.py before quantization.
- Pre-Tokenized Dataset Invalidation: Changing bin resolution in discrete_action.py
  invalidates pre-tokenized training shards and requires re-running dataset preprocessing.
```

3. Scenario failure diagnosis and 2D Bird's-Eye-View comparative overlays (`womd-failure-diagnosis`; expert seed skill). Aggregate development-split scores indicate *whether* an edit improved average accuracy, but obscure *why* specific traffic interactions succeeded or failed. To ground hypothesis generation in multi-agent geometry, the Researcher invokes the `womd-failure-diagnosis` skill through the read-only observability API (Listing 10). Given a candidate submission file, the diagnostic suite mines the worst-performing development-split scenarios and generates both structured textual reports and static 2D Bird's-Eye-View (BEV) comparative scenario plots (`.png` in global (X, Y) meter coordinates; Figure 4). Each 2D BEV plot overlays the high-definition roadgraph (lane centerlines, dashed road boundaries, and yellow crosswalk polygons), the contextual autonomous vehicle (`Ego (AV)` blue bounding box and solid blue `Ego GT Future`), the target evaluated actor (`Actor <id>` red bounding box

and dashed red ground-truth trajectory), the faint gray cloud of all $N = 512$ unaggregated raw trajectory proposals, and the $K = 6$ aggregated modes (`Modes 1--6 (P:...)`) shaded by confidence with endpoint index markers.

Listing 10: The `womd-failure-diagnosis/SKILL.md` specification used by the Researcher to mine development-split scenario failures and inspect static 2D BEV comparative trajectory overlays.

```
---
name: womd-failure-diagnosis
description: >-
  Guide for running scenario-level failure diagnostics and rendering static 2D
  Bird's-Eye-View (BEV) comparative PNG overlays from model runs to isolate
  specific post-processing failure modes and extract fine-grained diagnostic signals.
---

# Scenario Failure Diagnosis & 2D BEV Comparative Analysis

## 1. End-to-End Execution Workflow
1. Run Diagnostic Suite on Candidate Submission: Execute the diagnostic pipeline with
   --debug=true so textual reports and 2D BEV .png plots are written to /tmp/failure_mining/:
   ./scripts/diagnose_predictions.sh --submission_pb_file="${PB_PATH}" --debug=true

2. Inspect Master Summary & Select Target Scenarios:
   Read /tmp/failure_mining/<eval_id>/analysis_suite_summary.md to identify worst-scoring
   development-split scenarios, then inspect individual failure_report_<scenario_id>.md files
     and their
   paired 2D BEV Comparative Scenario plots (bev_base_<id>.png and bev_test_<id>.png).

## 2. Interpreting 2D BEV Comparative Plots & Mechanistic Failure Taxonomy
Each 2D BEV plot renders Global X/Y (meters) with Ego (AV) context (solid blue), Target Actor
GT (dashed red), the 512 raw proposal cloud (faint gray), and the 6 selected modes (Modes 1--6
with confidence P and endpoint index labels). Classify each failure into 1 of 5 root causes:
1. Over-Suppression: The faint gray 512-rollout cloud covers the dashed red Actor GT path,
   but all 6 selected modes miss it -> Suppression radius too wide or horizon blind spot.
2. Mode Drift / Centroid Pullback: Selected modes in the turn fan drift radially inward
   or lag behind the dashed red GT endpoint -> Unweighted K-Means coordinate pullback.
3. Redundant Mode Packing: Multiple numbered mode endpoints cluster tightly along a single
   branch while alternative branches in the gray proposal cloud go unrepresented.
4. Probability Mismatch / Bad Ranking: A mode closely tracks the dashed red Actor GT
   (e.g., Mode 3 or Mode 5) but receives lower confidence P than modes that veer off-lane.
5. Generator Failure: The entire faint gray cloud of 512 raw proposals misses the dashed
   red Actor GT -> STOP. Do NOT tune post-processing parameters on upstream generator misses.

## 3. Formulating the Output Hypothesis
Conclude your report with a <hypothesis_for_next_turn> block diagnosing the algorithmic root
cause (e.g., centroid drift or probability inversion) without prescribing raw scalar values.
```

G Additional Case-Study Details

The cases shown here start from the baseline MotionLM autoregressive checkpoint described in Appendix D. Scores reported within this appendix, including the Candidate A–C comparisons, are 80-shard development-split results used during search. The protected-split scores quoted in Section 4.4 were computed through the harness after the episode terminated, solely for reporting; they were never returned to agents or used for selection.

G.1 Case Study 1: Hybrid Discrete–Continuous Residual Refinement

Diagnostic trigger. Inspection of long-horizon training trajectories via `plot_curves` revealed a striking decoupling: sequence cross-entropy on discrete action tokens continued to decrease steadily, yet trajectory displacement error (minADE) plateaued completely after 400k steps. Analysis attributed this plateau to quantization in the $13 \times 13 = 169$ Verlet-action grid. With unit-spaced bins, the maximum within-bin quantization radius is $\sqrt{0.5^2 + 0.5^2} \approx 0.71$ grid units in action space; it is not a fixed 0.71 m displacement error per prediction step, because position error depends on subsequent Verlet integration.

Core mechanistic insight. Expanding the discrete vocabulary resolution (e.g., to $25 \times 25 = 625$ tokens) would increase the vocabulary-dependent output projection and softmax work approximately linearly with vocabulary size ($625/169 \approx 3.70$ times for those operations), without implying

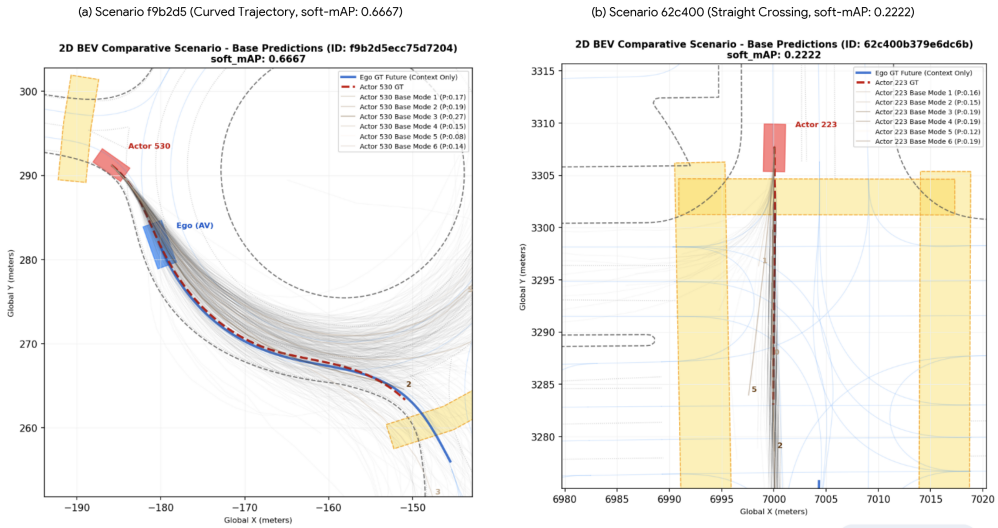

Figure 4: Representative 2D Bird's-Eye-View (BEV) comparative scenario overlay generated by the `womd-failure-diagnosis` skill. By inspecting how the $K = 6$ probability-labeled modes (`Modes 1--6`) and the underlying $N = 512$ raw proposal cloud (faint gray) align with the target actor's ground-truth trajectory (dashed red) and `Ego (AV)` context (solid blue), the Researcher distinguishes post-processing confidence inversions and mode drift from upstream generator misses.

the same increase in total decoding latency. Instead, DRIVELAB proposed a hybrid decomposition: retain the discrete 169-token classification head to model multimodal branching while branching a continuous sub-grid Verlet-action refinement head directly from the decoder embeddings.

Visual architecture and mathematical formulation. For an assigned discrete token bin with Verlet-action coordinate $\mathbf{a}_{\text{bin}}(\hat{a})$, the continuous sub-grid action residual target is

$$\Delta\mathbf{a}^* = \mathbf{a}_{\text{ground truth}} - \mathbf{a}_{\text{bin}}(\hat{a}). \tag{4}$$

As illustrated in Figure 5, the shared transformer decoder embedding $\mathbf{q} \in \mathbb{R}^{B\times N\times T\times D}$ feeds both the discrete classifier and the action-residual regressor. The corrected continuous actions are integrated by the existing Verlet decoder. The composite objective optimizes

$$\mathcal{L}_{\text{total}} = \mathcal{L}_{\text{CE}}(\mathbf{z}, a^*) + \lambda_{\text{ref}} \cdot \frac{1}{\sum M} \sum_{b,n,t} M_{b,n,t} \cdot \text{Smooth}_{L_1}(\Delta\mathbf{a}_{b,n,t} - \Delta\mathbf{a}^*_{b,n,t}), \tag{5}$$

where $\text{Smooth}_{L_1}(r) = 0.5r^2/\delta$ when $|r| < \delta$ and $|r| - 0.5\delta$ otherwise, with refinement weight $\lambda_{\text{ref}} = 0.25$. The threshold $\delta = 0.2$ is expressed in the normalized Verlet-action coordinates, not meters. Listing 11 details the core JAX implementation.

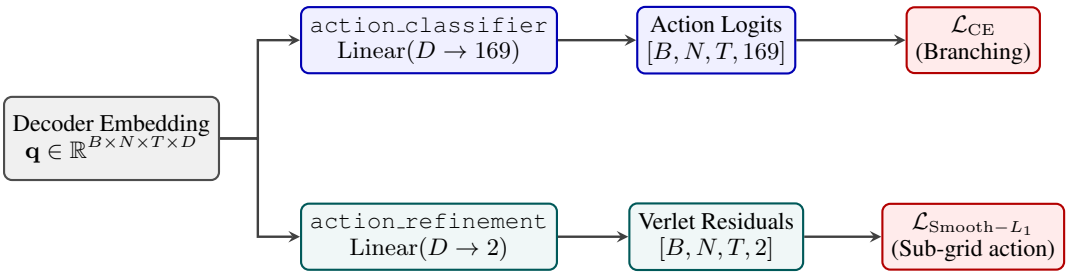

Figure 5: Hybrid discrete–continuous network architecture. Discrete action classification preserves multimodal trajectory branching while parallel continuous action regression reduces sub-grid quantization error before Verlet integration.

Listing 11: JAX implementation of hybrid continuous residual refinement.

```
# Discovered refinement loss in models/motionlm/prediction.py
base_loss, metrics = super().compute_loss(batch, action_logits)
action_x = (target_actions % 13).astype(jnp.float32) - 6.0
action_y = (target_actions // 13).astype(jnp.float32) - 6.0
discrete_verlet_action = jnp.stack([action_x, action_y], axis=-1)

gt_action_residuals = gt_verlet_actions - discrete_verlet_action
residual_diff = action_residuals - gt_action_residuals

huber_delta = 0.20
abs_diff = jnp.abs(residual_diff)
smooth_l1 = jnp.where(
    abs_diff < huber_delta,
    0.5 * jnp.square(residual_diff) / huber_delta,
    abs_diff - 0.5 * huber_delta)

valid_mask = batch['output/target_valid_mask'][..., None]
refinement_loss = jnp.sum(smooth_l1 * valid_mask) / jnp.maximum(jnp.sum(valid_mask), 1.0)
total_loss = base_loss + 0.25 * refinement_loss
metrics['refinement_loss'] = refinement_loss
```

G.2 Case Study 2: Multi-Horizon MMR Selection and Entropic Sinkhorn Optimal Transport

To illustrate an automated discovery that is completely independent of actor-class routing and maneuver-bucket priors, we trace the discovery of `MultiHorizonAdaptiveAggregator` (`multi_horizon_nms`), which reformulates how $N = 512$ autoregressive trajectory rollouts are aggregated into $K = 6$ multimodal predictions and confidence scores on the Waymo Open Motion Dataset (WOMD).

(1) Diagnostic Trigger from Agent-Visible Development Evidence. During development evaluations on the baseline MotionLM autoregressive checkpoint described in Appendix D, the Researcher used the expert-authored seed skill `womd-failure-diagnosis` to analyze scenario-level failure reports and horizon-sliced precision–recall curves across the 8.0 s prediction window ($T = 80$ steps at 10 Hz). Thus, the diagnostic evidence that motivated Case 2 came from a seed workflow available at episode initialization, not from an automatically distilled skill. Two structural defects in the baseline aggregator were identified:

1. **Single-Horizon Blind Spots at $t = 5.0$ s:** The baseline greedy cover computed pairwise Euclidean distances solely at zero-based step index $t = 49$ (5.0 s). In highway merges and intersections, rollouts that shared a lane centerline at 5.0 s before branching into adjacent lanes or exit ramps at 8.0 s ($t = 79$) satisfied $d_{49}(i, j) \leq \tau$ and were erroneously suppressed as duplicates.
2. **Centroid Pullback and Mode Collapse under K-Means:** Refining selected seeds via hard-assignment K-Means averaged fast outer-arc trajectories with slower interior rollouts. This systematically underestimated longitudinal speed (inflating 8.0 s endpoint lag) and allowed low-density secondary branches to collapse onto dominant lane-following modes.

(2) Hypothesis Proposed by the Researcher. The Researcher proposed replacing both stages of the baseline aggregator with a metric-aligned formulation:

1. **Multi-Horizon Velocity-Scaled Distance:** Because WOMD evaluates predictions across 3.0 s, 5.0 s, and 8.0 s (zero-based step indices $t \in \{29, 49, 79\}$), pairwise suppression normalizes trajectory displacement at each horizon by speed-adaptive radii $\tau_{49}(v_i) = \tau_0 + \gamma v_i$, with $\tau_{29} = 0.6\,\tau_{49}$ and $\tau_{79} = 1.5\,\tau_{49}$:

$$D_{\text{multi}}(i,j) = 0.20\,\frac{\|\mathbf{x}_i(29) - \mathbf{x}_j(29)\|_2}{\max(\tau_{29,i}, \tau_{29,j})} + 0.55\,\frac{\|\mathbf{x}_i(49) - \mathbf{x}_j(49)\|_2}{\max(\tau_{49,i}, \tau_{49,j})} + 0.25\,\frac{\|\mathbf{x}_i(79) - \mathbf{x}_j(79)\|_2}{\max(\tau_{79,i}, \tau_{79,j})}. \tag{6}$$

   Suppression uses $D_{\text{multi}}(i, j) \leq 1$. Because this is a weighted sum rather than a conjunction of three threshold tests, closeness at one horizon can offset greater separation at another.

For example, if $d_{29} = d_{49} = 0$ and $v = 0$, then $\tau_{49} = 1.5\,\text{m}$ and pairs separated by as much as $d_{79} = 9\,\text{m}$ still meet the aggregate threshold.

2. **Maximal Marginal Relevance (MMR) Mode Selection:** Instead of greedy probability cover alone, each slot $k \in \{1, \ldots, 6\}$ maximizes normalized uncovered probability mass $\tilde{C}(i)$ plus a repulsion bonus $\lambda_{\text{mmr}}\, \tilde{d}_{\min}(i)$ from already-selected modes, preventing late-horizon branch starvation.
3. **Entropic Sinkhorn Optimal Transport & Speed-Dependent Seed Blending:** Rollout mass $\mathbf{p} \in \Delta^{N-1}$ is coupled to target mode marginals $\mathbf{q} = (1-\alpha)\,\mathbf{p}_{\text{nms}} + \alpha\,(1/K)\mathbf{1}$ via entropic optimal transport ($\varepsilon \in [2.0, 2.5]$). The resulting Wasserstein barycenters $\mathbf{Y}_k$ are convexly blended with the raw kinematically feasible seed $\mathbf{X}_{s_k}$ using $\alpha_k(v_k) = \text{clip}(0.80 - 0.012\,v_k,\, 0.45,\, 0.80)$ as the seed weight. Thus, the seed contribution decreases as speed rises, while the clipped lower bound retains a minimum seed contribution of 0.45; the schedule is not an increasing high-speed anchor.

(3) Realized Implementation by the Coder. The Coder implemented `MultiHorizonAdaptiveAggregator` in `aggregation.py` and wired it into the post-processing pipeline. When initial prototypes invoked `jnp.linalg.solve` for trajectory smoothing, the Coder diagnosed a runtime failure on CPU evaluation workers (`NotFoundError: No FFI handler registered for lapack_sgetrf_ffi`) and replaced the unsupported solve with vectorized JAX operations. In Listing 12, MMR selection uses `jax.lax.scan`, while log-domain Sinkhorn scaling uses a fixed 100-iteration Python `for` loop whose body consists of vectorized JAX operations.

(4) Development-Side Result and Worked Selection Example. During search, three competing Researcher proposals were implemented by Coders and evaluated on the development split before one was promoted:

- *Candidate A (Strict Boolean Pareto Conjunction, `pareto_dominance`):* Required $(d_{29} \le \tau_{29}) \wedge (d_{49} \le \tau_{49}) \wedge (d_{79} \le \tau_{79})$. Almost no rollout pair satisfied all three bounds simultaneously, starving local density coverage and dropping Soft mAP to **0.4170**. The Manager recorded this "boolean conjunction trap" in the negative-results memory.
- *Candidate B (Unbalanced Optimal Transport with KL relaxation, $\rho = 1.0$):* Relaxing marginal constraints allowed secondary cluster weights to decay toward zero, merging distinct branches onto the dominant mode (**0.4242** Soft mAP on the 80-shard development set).
- *Candidate C (`multi_horizon_nms`, Promoted Winner):* Replaced the boolean conjunction with the continuous convex combination $D_{\text{multi}}(i,j) \le 1.0$ and balanced Sinkhorn marginals ($\alpha = 0.5$). On the 80-shard development split, Soft mAP reached **0.4538**.

(5) Independent Scope/Novelty Grading and Verification. Under our deterministic Scope rubric and blind human verification, the implementation received **Scope 2** (adds a new aggregation module within the sampling and post-processing stage) and **Novelty 2** (non-trivially adapts and combines established multi-horizon alignment, MMR selection, and entropic optimal transport techniques for motion forecasting). The aggregator is class-agnostic by design: every actor type executes the same branch with the same aggregation parameters. The reported Soft mAP gains therefore arise from the multi-horizon MMR and optimal-transport mechanics rather than class-specific hyperparameter tuning.

(6) Code Excerpt. Listing 12 shows the core JAX implementation of `MultiHorizonAdaptiveAggregator`.

Listing 12: Core multi-horizon MMR selection and log-domain Sinkhorn optimal transport refinement in `MultiHorizonAdaptiveAggregator`

```
def _run_single_agent(agent_traj: Array, agent_probs: Array, epsilon_agent: float):
  num_traj = agent_traj.shape[0]
  p = agent_probs / (jnp.sum(agent_probs) + 1e-12)

  # 1. Multi-horizon (3.0s, 5.0s, 8.0s) velocity-adaptive normalized distance
  traj_t29, traj_t49, traj_t79 = agent_traj[:, 29], agent_traj[:, 49], agent_traj[:, 79]
  d29 = jnp.linalg.norm(traj_t29[:, None, :] - traj_t29[None, :, :], axis=-1)
  d49 = jnp.linalg.norm(traj_t49[:, None, :] - traj_t49[None, :, :], axis=-1)
  d79 = jnp.linalg.norm(traj_t79[:, None, :] - traj_t79[None, :, :], axis=-1)

  speeds = jnp.linalg.norm(agent_traj[:, -1] - agent_traj[:, 0], axis=-1) / 8.0
  thresh_49 = 1.5 + 0.12 * speeds
  thresh_mat_49 = jnp.maximum(thresh_49[:, None], thresh_49[None, :])
  norm_dist = (
      0.20 * (d29 / (0.6 * thresh_mat_49 + 1e-6))
      + 0.55 * (d49 / (thresh_mat_49 + 1e-6))
      + 0.25 * (d79 / (1.5 * thresh_mat_49 + 1e-6))
  )
  near_mask = norm_dist <= 1.0

  # 2. Maximal Marginal Relevance (MMR) scan for K=6 diverse mode seeds
  def _mmr_step(state, _):
    avail_probs, sel_mask, min_dist = state
    cov_scores = jnp.sum(near_mask * avail_probs[None, :], axis=1)
    mmr_score = (cov_scores / (jnp.max(cov_scores) + 1e-6)) + (
        0.25 * (min_dist / (jnp.max(min_dist) + 1e-6))
    )
    mmr_score = jnp.where(sel_mask, -jnp.inf, mmr_score - jnp.arange(num_traj) * 1e-6)
    best_idx = jnp.argmax(mmr_score)
    p_nms_k = jnp.sum(near_mask[best_idx] * avail_probs)
    return (
        jnp.where(near_mask[best_idx], 0.0, avail_probs),
        sel_mask.at[best_idx].set(True),
        jnp.minimum(min_dist, norm_dist[best_idx]),
    ), (best_idx, p_nms_k)

  _, (selected_indices, p_nms) = jax.lax.scan(
      _mmr_step, (p, jnp.zeros(num_traj, bool), jnp.full(num_traj, 1e6)), None, length=6
  )

  # 3. Entropic Sinkhorn Optimal Transport & speed-dependent seed blending
  p_nms = p_nms / (jnp.sum(p_nms) + 1e-12)
  q = 0.5 * (1.0 / 6.0) + 0.5 * p_nms  # Balanced marginal prior
  seed_traj = agent_traj[selected_indices]
  cost_matrix = jnp.mean(jnp.linalg.norm(
      agent_traj[:, None, :, :] - seed_traj[None, :, :, :], axis=-1
  ), axis=-1)
  log_K = -cost_matrix / epsilon_agent
  log_u, log_v = jnp.zeros(num_traj), jnp.zeros(6)
  for _ in range(100):
    log_u = jnp.log(p + 1e-12) - jax.scipy.special.logsumexp(log_K + log_v[None, :], axis=1)
    log_v = jnp.log(q + 1e-12) - jax.scipy.special.logsumexp(log_K + log_u[:, None], axis=0)
  pi = jnp.exp(log_K + log_u[:, None] + log_v[None, :])
  barycenters = jnp.einsum('nk,ntd->ktd', pi, agent_traj) / (jnp.sum(pi, axis=0)[:, None, None
     ] + 1e-12)
  alpha_k = jnp.clip(0.80 - 0.012 * speeds[selected_indices], 0.45, 0.80)
  centroids = alpha_k[:, None, None] * seed_traj + (1.0 - alpha_k[:, None, None]) *
     barycenters
```