WEB2LONGTRAJ: SCALING LONG-HORIZON AGENT TRAJECTORIES FROM NATURAL WEB CORPORA FOR MID-TRAINING

Anonymous authors
Paper under double-blind review

ABSTRACT

Long-horizon agent trajectories are expensive to obtain because most existing pipelines synthesize them in manually engineered executable environments. We present **Web2LongTraj**, a corpus-first pipeline that mines natural web corpora for real multi-step problem-solving records and compiles them into source-grounded agent trajectories, with actions and outcomes constrained by the source and observations audited for machine-output form. Screening 317 million web records, Web2LongTraj yields 1.15 million audited trajectories. To scale these short episodes into long-horizon supervision, we introduce two operators that grow them along the two axes on which real sessions become long. **AnchorStretch** extends dependency distances within a trajectory: it first identifies an early premise and a later step that depends on it, then inserts topically similar audited episodes between them. **ArtifactStitch** composes multi-task sessions by walking a graph that connects trajectories through shared artifacts and compatible working contexts. By recombining the 1.90B-token audited pool, the two operators generate 10.6B long-context tokens, a 5.6-fold expansion without building new task environments. Mid-training Olmo-3-7B and Olmo-3-32B on Web2LongTraj with a standard next-token objective yields consistent gains over equal-budget baselines on tool calling, multi-step interaction, long-horizon memory, and coding benchmarks under both Instruct and Think post-training recipes, while preserving general capability. On Olmo-3-7B, the full Web2LongTraj recipe improves all eight benchmarks by an average of 4.7 points under Instruct SFT and 9.9 points under Think SFT; its five-benchmark general-capability mean also increases by 0.4 and 3.3 points, respectively. These results establish corpus-first trajectory construction as a practical complement to environment-first synthesis for training long-horizon agents.

1 INTRODUCTION

Large language models (LLMs) are no longer expected to solve only short question-answering tasks. They are increasingly deployed as agents that assist humans with long-horizon tasks, such as debugging, web interaction, and multi-step software workflows. As sessions lengthen, early constraints can be forgotten, and assumptions from completed tasks can interfere with new requests. In these settings, success depends not only on reasoning ability, but also on maintaining context across long trajectories, recovering earlier decisions after many intermediate steps, and distinguishing among multiple related tasks within the same session (Patil et al., 2025; Wu et al., 2024; Zhao et al., 2026). As a result, long-context and long-horizon trajectory modeling have become central to improving agentic LLMs.

A common approach to training such agents is to construct executable environments and synthesize trajectories within them (Qin et al., 2024; Prabhakar et al., 2026; Xu et al., 2025b). While this paradigm provides controllable supervision, it is costly to build and difficult to scale. Each new domain often requires custom tools, environment design, and verification pipelines, making data generation expensive in both engineering effort and compute. Moreover, environment-based synthesis is naturally limited by the scope of the designed environment, which restricts the diversity of tasks, failure modes, and interaction patterns that the model can observe during training.

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.

In fact, many natural web records already contain the implicit structure of agent interactions. Forum discussions, issue threads, notebooks, and execution logs often record a user posing a problem, actions being taken, outputs being observed, and the process being iteratively refined. Their coverage across domains provides a broad source of diverse tasks, tools, and failure modes. For records that document these interactions, the procedural content is already present; what is missing is the explicit agent form. We therefore present **Web2LongTraj**, a corpus-first pipeline that constructs source-grounded long-horizon agent trajectories from natural web records (Figure 1). In its first stage, Web2LongTraj screens 317 million records for documented multi-step problem-solving processes and reconstructs selected records into short ReAct-style trajectories with explicit reasoning, actions, and observations (Yao et al., 2022). This screening and reconstruction process yields 1.15 million audited short trajectories while keeping actions and observations faithful to the source evidence.

However, these trajectories mined from web corpora are typically short, averaging 1,657 tokens and 3.5 observations. They capture individual tasks but provide limited supervision for recovering an earlier dependency after substantial intervening work or managing several related tasks in one session. In its second stage, Web2LongTraj introduces two expansion operators that grow these short episodes along the two axes on which real sessions become long.

AnchorStretch extends dependency distances within a trajectory. It models the case where a user returns to an earlier thread of work after a long detour, or a later review reveals an omission that requires revisiting an earlier decision. The operator first identifies an early premise and a later step that depends on it, with an independent reviewer checking that the continuation requires the earlier information. Drawing on long-context synthesis with retrieved distractors (Gao et al., 2025), it then inserts complete, topically similar audited trajectories between the premise and its continuation. Their topical similarity creates semantic interference: the model cannot rely on a change of topic to locate the main trajectory and must correctly recover the early premise across the intervening long context.

ArtifactStitch composes multi-task sessions around shared artifacts. It models a user working through several related tasks within one project, where successive requests refer to overlapping files and tools while having distinct goals. ArtifactStitch builds a graph over the audited trajectories using shared artifacts, environment compatibility, and semantic relatedness, then walks the graph to compose sessions under a shared tool declaration. Because adjacent tasks share vocabulary and tools, the model must identify task boundaries from requests and interaction structure. It must keep completed goals separate from new requests while preserving shared context for the next task.

AnchorStretch and ArtifactStitch transform short mined trajectories into longer, more challenging sequences that better reflect real agent workloads. Together, they teach models to recover earlier information, decide when it applies, and reuse shared context without letting completed goals or task-specific assumptions interfere with new requests. Recombining the 1.90B-token audited pool produces 10.6B long-context tokens without new task environments.

We mid-train Olmo-3-7B and Olmo-3-32B (Olmo et al., 2025) on Web2LongTraj with a standard next-token objective over all tokens to expose the base model to tool interactions and long-range dependencies before supervised fine-tuning (SFT). Across both model sizes and both Instruct and Think SFT, the full recipe consistently improves tool calling, multi-step interaction, long-horizon memory, and coding while preserving general capability. Under Instruct and Think SFT, respectively, the eight-benchmark average improves by $+4.7$ and $+9.9$ points on Olmo-3-7B and by $+3.4$ and $+5.8$ on Olmo-3-32B. Our main contributions are as follows:

- **A corpus-first trajectory construction pipeline.** Web2LongTraj converts natural web records into source-grounded ReAct interactions and audits observations for machine-output form.
- **AnchorStretch and ArtifactStitch for richer long-context sessions.** We introduce two operators: AnchorStretch, which separates earlier premises from dependent continuations with complete, topically similar agent trajectories to extend dependency distances to 227K tokens; and ArtifactStitch, which uses graph-based composition over shared artifacts and compatible working contexts to produce 13,164 multi-task sessions under shared tool declarations.
- **Controlled evidence for each design choice.** Equal-budget comparisons validate the gains across both model sizes and SFT settings, while ablations show that audited agent form, explicit dependencies, and shared working context each contribute.

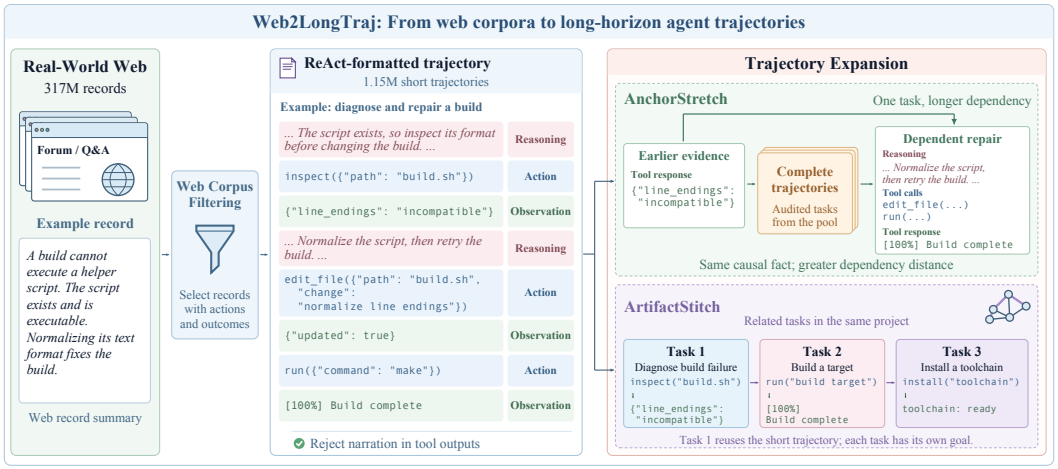


Figure 1: Overview of Web2LongTraj. Web records are reconstructed into audited short trajectories; AnchorStretch separates a premise from its dependent continuation, while ArtifactStitch composes related trajectories into multi-task sessions. Colors denote reasoning, tool calls, and tool responses; the example is schematic.

2 METHOD

Let $\tau = (u, r_1, a_1, o_1, \ldots, r_T, a_T, o_T, y)$ denote an agent trajectory with user request u, reasoning r_t, action a_t, observation o_t, and final response y. Web2LongTraj builds the mid-training mixture in two stages (Figure 1): it first mines and compiles records into audited short trajectories (Section 2.1), then expands them with AnchorStretch (Section 2.2) and ArtifactStitch (Section 2.3).

2.1 MINING SHORT TRAJECTORIES FROM WEB CORPORA

A useful web record must contain both an action and its returned outcome. This structure is common in community discussions, where users often report commands together with their outputs. We process the crawl through three increasingly strict gates (Figure 2).

Gate 1: is this record worth compiling? Knowledge quality alone does not imply agentic value. Qwen3.6-35B-A3B scores each record in $\{0, \ldots, 5\}$ on six dimensions $\mathbf{s}(d)$: procedurality, actionability, observability, decision density, trajectory convertibility, and knowledge quality. The Agentic Readiness Score (ARS) combines these scores with a structural ceiling:

$$\mathrm{ARS}(d) = \min\big\{\, A(\mathbf{s}(d)),\ C(d) \,\big\}, \tag{1}$$

where A is the level that the six dimension scores support, from 0 for unusable text to 5 for a process that also reports observed outcomes, logs, or error fixes, and $C(d) \in \{1, 2, 3, 5\}$ is a structural ceiling imposed by the rubric's hard gates. The ceiling prevents high scores on other dimensions from compensating for structural defects; its complete rules are given in Section A. The model applies the rubric and returns $\mathrm{ARS}(d)$ in the same response. Records with $\mathrm{ARS} \geq 4$ form the seed pool, corresponding to 2.1% of 317.4M records. LLM rubric scoring is established for selecting pretraining data at web scale (Penedo et al., 2024). Section I.1 provides the full prompt.

Gate 2: is a trajectory really there? We represent each record using the ReAct protocol, which explicitly alternates reasoning, actions, and returned observations (Yao et al., 2022). For eligible records that already contain these elements, a GLM-5.2-FP8 compiler reconstructs them into a typed sequence of `think`, `action`, `observation`, `say`, and `final` steps, which Python renders into the training protocol. This stage retains 28.4% of the seed pool, converting 6.68M seeds into 1,893,342 trajectories with averages of 1,657 tokens and 3.52 observations. Section A gives the schema inference and mechanical constraints, and Section I.2 provides the full prompt.

Gate 3: is every observation really machine output? The dangerous failure is a human summary dressed as a tool response. Source support alone does not guarantee valid observation form: human narration may accurately describe an outcome without being literal machine output. A GLM-5.2-FP8

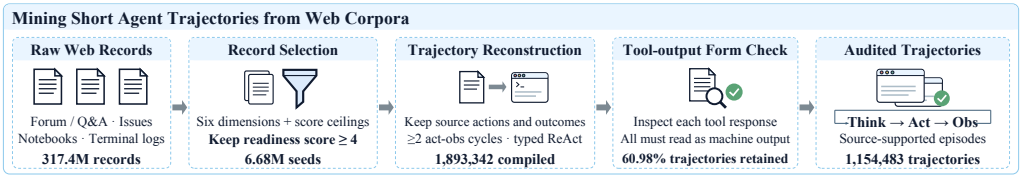


Figure 2: Construction of SeedTraj through readiness scoring, source-grounded multi-step reconstruction, and rejection of narrated observations.

judge sees the observation text alone and decides whether it reads as literal output from a shell, compiler, API, database or browser, and a trajectory is admitted only if every one of its observations passes. This decision has to be semantic rather than lexical. The judge marks 22.26% of 6.67M observations as narrated. Because a single narrated observation rejects its whole trajectory, 39.02% of trajectories are rejected, leaving 1,154,483 trajectories containing 1.90B tokens. We call this audited pool *SeedTraj*. It directly supplies the short-trajectory component of Stage 2 and serves as the source pool for AnchorStretch and ArtifactStitch. Section I.3 provides the full prompt.

2.2 AnchorStretch: Stretching a Trajectory's Own Dependency

An admitted trajectory contains one task in about 1.6K tokens. AnchorStretch converts it into long-horizon supervision by splitting the trajectory at an existing dependency and inserting complete hard-negative trajectories between the two parts. The dependency itself is unchanged; only its distance increases, from roughly 1.6K tokens to 52K, 109K, or 227K (Figure 3).

Split where the dependency is, not at a fixed length. Write an admitted trajectory as balanced blocks $b_1, \ldots, b_n$. We identify an early anchor a and a later continuation beginning at b_k whose reasoning depends on information established before k, giving $\tau = p \parallel s$ with $p = b_{1:k-1}$ and $s = b_{k:n}$. Both parts are copied byte for byte. The extraction model scores its proposed dependency, but admission is decided by an independent reviewer on its own scores rather than on this one; both are GLM-5.2-FP8 with separate prompts. The reviewer retains 23.2% of candidates for which s genuinely requires the anchor, cannot be answered from s alone, and remains coherent after interruption, and every admitted split also passes mechanical integrity checks (Section A). This extractive construction introduces no new action, observation, or outcome. A complementary *probe* construction appends a follow-up request requiring an early value from the source trajectory and is ablated separately (Section B). Section I.4 provides the four prompts used for split construction and review.

Stretch the distance with complete trajectories. For a main pair (p, s), we sample M distractor trajectories and construct

$$x = h \parallel p \parallel \tau_1^- \parallel \cdots \parallel \tau_M^- \parallel c \parallel s, \tag{2}$$

where h is the system and tool header and c is a resume cue. Each distractor is a complete trajectory that passes the same observation audit as the main. The header declares every tool used anywhere in the sample; otherwise calls to undeclared tools would provide a trivial cue for locating distractors. We prefer topically similar distractors because unrelated trajectories create obvious topic boundaries, while similar trajectories create semantic interference and require the model to correctly recover the earlier premise from the intervening context.

We compare three retrieval policies: **random** sampling across tasks, **lexical** retrieval over shared task-signature tokens, taken from the first user request and the names of the called tools, using MinHash and locality-sensitive hashing (Broder, 1997), and **hybrid** retrieval that additionally uses nearest neighbours from Qwen3-Embedding-8B (Zhang et al., 2025). Lexical retrieval is limited by the number of similar trajectories available in the corpus, so its share falls from 32.1% at 64K to 15.3% at 256K as more distractor slots are required, while semantic retrieval remains near 30% at every length. This makes semantic retrieval important for maintaining similar distractors as the context grows. All other factors, including the main trajectory, anchor, continuation, cue, header, token budget, and seed, are held fixed across the three policies (Section A, Table 4).

Check that the dependency is real. Long context alone does not guarantee that earlier information is necessary. For a candidate (π, σ), where σ is the later portion to be produced and π is its required earlier context, let $V \in \{0, 1\}$ indicate whether a GLM-5.2-FP8 grader judges σ to be produced

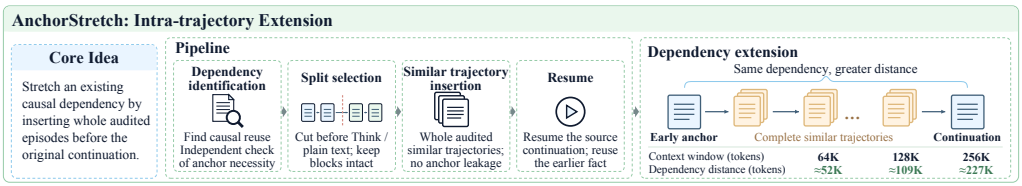


Figure 3: AnchorStretch extends an existing dependency by inserting complete, audited, topically similar trajectories before its continuation. An independent review checks whether the continuation requires the earlier anchor.

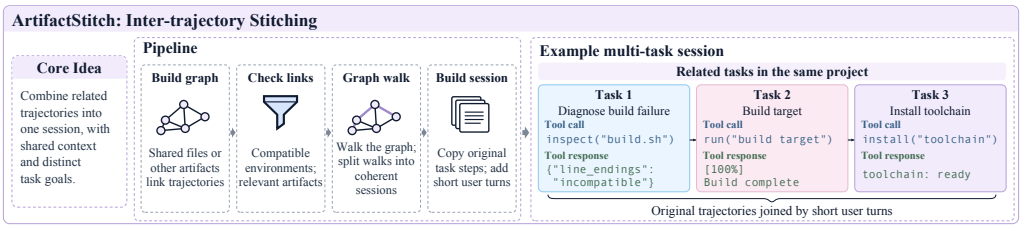


Figure 4: ArtifactStitch filters artifact-based candidate links by compatibility and semantic support, then converts bounded graph walks into multi-task sessions with preserved task bodies and a merged tool declaration. The example is schematic.

correctly. We measure

$$\Delta = \mathbb{E}\big[V \mid \text{context contains } \pi\big] - \mathbb{E}\big[V \mid \pi \text{ removed}\big]. \tag{3}$$

Each item is graded once with π and once with π removed. AnchorStretch splits obtain $\Delta \approx +0.60$, meaning that removing the earlier premise reduces grader pass rate by about 60 percentage points on the same items. This test verifies that the constructed sample requires the long-range dependency rather than merely containing long or topically related context.

2.3 ARTIFACTSTITCH: COMPOSING RELATED TRAJECTORIES INTO MULTI-TASK SESSIONS

Real agent sessions often contain several related tasks within the same project. A user may configure a service, diagnose why it fails, and then verify the fix while continuing to work with the same files, tools, and environment. ArtifactStitch constructs this type of multi-task supervision by combining mined trajectories that share a working context.

An ArtifactStitch sample concatenates several related trajectories under one tool declaration, with a short templated user turn between consecutive tasks. The trajectory bodies are copied byte for byte, and each session uses a merged declaration covering the tools used by all member trajectories. Because adjacent tasks share files, tools, and vocabulary while pursuing different goals, the model must distinguish task boundaries, preserve relevant shared context, and avoid carrying assumptions from a completed task into the next request (Figure 4).

The graph: finding trajectories from a shared working context. We construct a sparse graph over the admitted trajectories, where each node is a trajectory. For each artifact, namely a file name with its extension, we first identify trajectories that mention it and form candidate connections between those that could plausibly belong to the same working environment. We avoid connecting trajectories from incompatible environments, such as Windows and Linux or different software stacks, and reject session orderings with an implausible backward jump in the workflow; keyword rules assign these stack, operating-system, and workflow-phase tags (Section C). Because generic names such as `test.py` also appear in unrelated records, we keep an artifact only when the pairs it links are also embedding neighbors far more often than chance; for retained artifacts, they are at least 2,400 times as likely as random pairs to be neighbors (Section C). We merge the retained connections into a sparse graph, take bounded walks over it, and split the walks into coherent sessions.

ArtifactStitch and AnchorStretch enter the mixture as separate components. AnchorStretch stretches a dependency within one task, whereas ArtifactStitch exposes the model to transitions between multiple related tasks in the same context.

3 EXPERIMENTS

3.1 SETUP

Model, objective, and budget. Most experiments continue from the Olmo-3-7B Stage 1 checkpoint, which is the checkpoint after pretraining, through Stage 2 and Stage 3 with next-token prediction and no loss masking (Olmo et al., 2025), using a fixed 60B-token budget. Stage 2 trains on 40B tokens at a sequence length of 8,192, including 1.90B SeedTraj tokens; Stage 3 trains on 20B tokens across 64K, 128K, and 256K contexts with YaRN (Peng et al., 2024), half long-context data and half official Olmo 3 short-context data. We also add our data to the full 150B-token Olmo-3-7B recipe and evaluate it at larger scale with Olmo-3-32B. Section D gives the full training and packing details.

Data recipes. We use five recurring mid-training recipes. Because the released Olmo 3 models were trained with contexts up to 64K (Olmo et al., 2025), we construct the **Olmo 3 long data** control by sampling the official Dolma 3 Longmino mixture and packing equal token budgets at 64K, 128K, and 256K to match our training lengths. **SeedTraj** replaces a token-matched portion of Stage 2 with audited short trajectories; **SeedTraj + AnchorStretch** additionally replaces the long-context portion of Stage 3 with dependency-stretched trajectories; **SeedTraj + ArtifactStitch** instead places graph-mined multi-task sessions in the 256K sub-stage, with Olmo 3 long data in the remaining long-context slots; and **Web2LongTraj** combines all three components. Unless stated otherwise, AnchorStretch uses hybrid retrieval. Except in the additive experiment, our data replaces equal-sized subsets of Olmo 3 long data, while each controlled comparison keeps the checkpoint, training budget, and evaluation protocol fixed. Section D gives the packing details.

Post-training. We evaluate every mid-trained checkpoint after both Instruct and Think supervised fine-tuning (SFT). The Instruct setting uses Olmo 3's released Instruct SFT (Olmo et al., 2025). Because the released Think SFT contains no agent trajectories, we use our standardized Think SFT, in which agentic trajectories contribute 10.0% of unique trainable tokens and general reasoning data contributes the remaining 90.0%. Within each setting, every configuration uses the same SFT data and training budget, ensuring that performance differences arise only from the mid-training data recipe. Section D gives the sources, mixture weights, formatting, and training details.

Evaluation. We use six agentic benchmarks and two code benchmarks, together with five general-capability benchmarks described in Section 3.5. BFCL-V4 (BFCL; Patil et al., 2025) measures tool selection and function calling. InterCode-Bash (IC-Bash; Yang et al., 2023) and τ^2-bench (Barres et al., 2025) test multi-step interaction with execution feedback or shared environment state. LongMemEval (Wu et al., 2024) and AMA-Bench (AMA; Zhao et al., 2026) measure recovery from long interaction histories, while CL-bench (CL; Dou et al., 2026) measures long-context task completion. MBPP+ (Liu et al., 2023) and BigCodeBench (BCB; Zhuo et al., 2025) measure coding transfer. We evaluate the complete official test sets. Absolute scores across Instruct and Think SFT are not directly comparable because they use different post-training mixtures and training budgets. Section E gives all benchmark, metric, decoding, and judging details.

3.2 MAIN RESULTS AND ABLATIONS

Table 1 compares the five recipes above on Olmo-3-7B at a fixed 60B-token budget. Its indented gray rows are the design ablations analyzed below.

The full Web2LongTraj recipe improves all eight benchmarks over Olmo 3 long data under both SFT settings, with average gains of $+4.7$ points under Instruct SFT and $+9.9$ under Think SFT. The component rows show that the two long-horizon operators are complementary: AnchorStretch helps most on tool calling and long-history memory, including $+6.1$ on LongMemEval under Think SFT, while ArtifactStitch helps most on the multi-turn tasks of τ^2-bench. Among the five main recipes, the full recipe attains the best score, including one tie, in 15 of the 16 comparisons across the two SFT settings. The additive comparison, which appends our corpus without removing control data, is reported in Section F.

Agent form is worth more than the content. The two gray rows under SeedTraj isolate the two transformations that produce it. Both use the same source records and token count as SeedTraj, and Stage 3 uses Olmo 3 long data throughout. The *raw web data* row keeps the source text unchanged, and the *w/o observation audit* row converts it to trajectories but skips the observation audit; SeedTraj

Table 1: Olmo-3-7B main results and design ablations at a fixed 60B-token budget. Indented gray rows change one choice in the preceding recipe; bold marks the best main recipe.

Data Recipe	BFCL	IC-Bash	τ^2-bench	LongMemEval	AMA	CL	MBPP+	BCB
Instruct SFT: fixed 60B budget								
Olmo 3 long data	23.9	75.8	9.0	23.3	24.6	20.3	45.2	24.5
SeedTraj	25.4	77.6	10.8	28.2	25.2	22.0	47.1	25.4
raw web data instead of trajectories	24.4	77.4	9.7	22.6	21.8	21.6	46.0	25.2
w/o observation audit	24.0	76.2	9.4	20.8	24.9	20.1	46.3	24.5
SeedTraj + AnchorStretch	29.1	80.2	11.0	29.9	27.0	23.7	50.8	26.8
w/o dependency split	27.6	78.9	10.4	20.1	25.6	21.7	49.5	25.2
random instead of hybrid retrieval	27.1	77.0	10.8	24.9	25.2	21.7	49.7	25.2
lexical instead of hybrid retrieval	27.2	78.0	10.3	20.8	24.9	21.6	47.1	27.5
SeedTraj + ArtifactStitch	25.9	79.5	**12.9**	30.1	26.8	22.1	49.7	26.3
unrelated sessions instead of graph walks	24.8	77.3	12.6	24.4	26.2	22.6	48.9	25.1
Web2LongTraj	**29.5**	**80.6**	12.6	**30.5**	**27.8**	**23.8**	**52.1**	**27.5**
Think SFT: fixed 60B budget								
Olmo 3 long data	12.6	60.9	17.7	21.7	12.6	14.5	24.1	19.0
SeedTraj	23.0	68.9	19.7	26.5	16.6	15.8	33.3	21.2
raw web data instead of trajectories	21.2	67.6	17.4	24.3	14.7	15.3	30.7	20.3
w/o observation audit	20.6	63.5	16.9	24.0	15.2	14.6	31.7	20.0
SeedTraj + AnchorStretch	24.3	70.0	19.8	32.6	20.9	16.3	**36.0**	21.3
w/o dependency split	23.2	67.3	21.0	24.5	19.0	14.3	31.5	20.9
random instead of hybrid retrieval	21.6	68.5	19.4	26.8	16.6	15.6	30.2	20.0
lexical instead of hybrid retrieval	23.6	69.0	18.7	28.6	24.0	14.6	31.7	20.4
SeedTraj + ArtifactStitch	23.4	69.0	24.0	26.1	19.0	15.6	35.4	21.3
unrelated sessions instead of graph walks	24.1	60.2	21.2	27.1	17.2	15.1	31.7	20.2
Web2LongTraj	**25.3**	**78.1**	**27.2**	**33.9**	**21.3**	**17.2**	**36.0**	**23.4**

applies both compilation and auditing. Agent form with audited observations turns web content into effective supervision. SeedTraj outperforms raw web data on all eight benchmarks under both SFT settings. The observation audit is key: it improves every benchmark over un-audited trajectories.

Explicit dependencies and hybrid retrieval improve long-horizon supervision. We isolate two properties of AnchorStretch using the gray rows under SeedTraj + AnchorStretch. The *w/o dependency split* row uses the same member trajectories in the same order but concatenates them as complete episodes, whereas SeedTraj + AnchorStretch inserts the same intervening episodes between an anchor and its dependent continuation. The two retrieval rows keep the split and change only how the intervening episodes are selected, replacing hybrid retrieval with random or lexical retrieval. Long context helps when it carries a dependency. Splitting at the dependency raises LongMemEval by 9.8 points under Instruct SFT and 8.1 under Think SFT, whereas the unsplit sequences score below the SeedTraj control on this benchmark. Hybrid retrieval, which keeps distractors topically similar as the context grows, is the best of the three retrieval policies on seven of eight benchmarks under each SFT setting and outperforms random distractors on all eight. Dependency admission matters as well: admitting splits through an independent reviewer instead of the construction model's own score improves all eight benchmarks under both SFT settings (Section B).

Shared working context improves multi-task sessions. The gray *unrelated sessions* row tests whether the gain of ArtifactStitch comes from multi-task formatting alone or from the shared working context recovered by the graph. It packs randomly grouped trajectories into sessions of the same size and places them in the 256K slot that ArtifactStitch fills with graph-linked sessions, so the comparison isolates task relatedness while holding the multi-task format fixed. Graph-linked sessions outperform unrelated sessions on IC-Bash, τ^2-bench, AMA, MBPP+, and BCB under both SFT settings, showing the value of connecting tasks through shared artifacts and working context.

3.3 AnchorStretch Remains Effective up to 256K Context

Stage 3 extends the context window cumulatively to 64K, 128K, and 256K. Because adding a sub-stage also adds training tokens, scores cannot be compared directly across window sizes. At each maximum window, we therefore compare the SeedTraj + AnchorStretch recipe against a SeedTraj control trained for exactly the same number of tokens. The valid comparison is within each pair; the change in that paired gap shows whether longer dependency distances add value (Figure 5).

The SeedTraj + AnchorStretch recipe outperforms its token-matched SeedTraj control on every benchmark at 64K, 128K, and 256K under both SFT settings (Table 7). Its benefit persists across

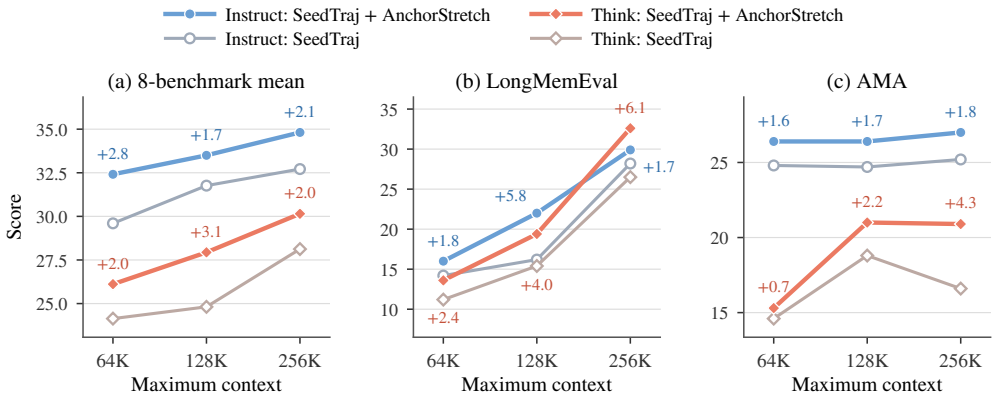


Figure 5: Context-length scaling for AnchorStretch. Labels show gains over token-matched SeedTraj controls; windows are cumulative and budgets differ across windows. Table 7 reports all metrics.

windows rather than being tied to one training length. Under Think SFT the advantage widens: on LongMemEval from +2.4 at 64K to +4.0 at 128K and +6.1 at 256K, and on AMA from +0.7 to +2.2 and +4.3. This shows that the method continues to exploit longer dependency distances.

3.4 Web2LongTraj Improves Across Model Scales

We repeat the five main recipes from Table 1 on Olmo-3-32B. All runs continue from the same Olmo-3-32B pretraining checkpoint and use the official Olmo-3-32B hyperparameters (Olmo et al., 2025): 20B tokens in Stage 2 and 20B in Stage 3. The SeedTraj share remains 4.7%, and AnchorStretch uses the hybrid policy selected on Olmo-3-7B. The recipe names have the same meaning as in the Olmo-3-7B table, although the stage budgets follow the Olmo-3-32B recipe.

Table 2: Olmo-3-32B results for the five main recipes, using 20B tokens in both Stage 2 and Stage 3.

Data Recipe	BFCL	IC-Bash	τ^2-bench	LongMemEval	AMA	CL	MBPP+	BCB
Instruct SFT								
Olmo 3 long data	31.3	81.1	14.7	14.4	31.9	28.5	56.3	36.8
SeedTraj	31.7	81.7	14.0	19.6	32.9	28.4	55.8	37.0
SeedTraj + AnchorStretch	32.1	84.0	14.9	23.4	33.0	29.4	58.2	**37.4**
SeedTraj + ArtifactStitch	32.7	83.2	14.2	25.4	32.0	29.4	56.9	36.1
Web2LongTraj	**33.2**	**84.8**	**16.6**	**27.4**	**33.3**	**30.2**	**59.5**	37.3
Think SFT								
Olmo 3 long data	27.2	70.8	27.4	37.0	20.8	27.0	46.0	34.7
SeedTraj	28.4	77.4	28.1	28.2	20.8	27.4	51.3	36.1
SeedTraj + AnchorStretch	29.7	79.2	29.1	**44.0**	24.9	27.8	52.6	**36.2**
SeedTraj + ArtifactStitch	29.6	75.4	28.8	38.0	25.0	26.6	51.1	34.6
Web2LongTraj	**30.8**	**80.0**	**38.8**	43.2	**27.9**	**28.2**	**52.9**	35.7

The gains are not specific to Olmo-3-7B. On Olmo-3-32B, the full Web2LongTraj recipe improves all eight benchmarks over Olmo 3 long data under both SFT settings, averaging +3.4 points under Instruct SFT and +5.8 under Think SFT. Its strong overall performance shows that the corpus-first pipeline scales with model size.

3.5 Web2LongTraj Preserves General Capabilities

We test whether the agentic gains trade off general knowledge, reasoning, or instruction following. MMLU measures broad academic knowledge (Hendrycks et al., 2020); MMLU-Pro and GPQA-Diamond stress harder reasoning and specialized scientific knowledge (Wang et al., 2024; Rein et al., 2023); BIG-Bench Hard (BBH) collects 23 challenging BIG-Bench tasks that require multi-step reasoning (Suzgun et al., 2023); and IFEval measures compliance with verifiable instructions (Zhou et al., 2023). We evaluate the five fixed-budget recipes from Table 1 under both SFT settings. Table 3 shows the primary comparison, while Section H reports the full five-recipe results and analysis.

Table 3: General-capability results for the primary fixed-budget Olmo-3-7B comparison; each SFT block uses the same checkpoint, 60B-token budget, and post-training recipe.

Data Recipe	MMLU	MMLU-Pro	GPQA-Diamond	BBH	IFEval
Instruct SFT					
Olmo 3 long data	**59.7**	31.7	30.3	41.6	**80.4**
Web2LongTraj	58.8	**32.1**	**30.8**	**43.8**	80.0
Think SFT					
Olmo 3 long data	68.5	46.3	26.8	54.2	64.7
Web2LongTraj	**69.0**	**47.0**	**30.3**	**63.5**	**67.3**

Compared with Olmo 3 long data, Web2LongTraj raises the five-benchmark mean by $+0.4$ points under Instruct SFT and $+3.3$ under Think SFT. It improves three of five benchmarks in the former setting and all five in the latter, showing that general capability is preserved overall and improves on average. Section H provides the full five-recipe results and analysis.

4 Related Work

Agent trajectory construction. Most tool-use datasets are built around predefined interfaces or executable environments. ToolLLM constructs function-calling supervision from large API collections (Qin et al., 2024); APIGen-MT generates task blueprints and simulates multi-turn interaction over executable APIs (Prabhakar et al., 2026); TOUCAN derives tool-use data from MCP servers (Xu et al., 2025b); and Magnet represents multi-tool workflows as graph-structured function paths before converting them into dialogues (Yin et al., 2025). Some work connects natural documents to agent behavior: AgentInstruct conditions agentic flows on raw documents (Mitra et al., 2024), while AgentTrek replays web tutorials in GUI environments so that the described actions are executed (Xu et al., 2025a). MidTool and GEM are the closest works that derive agentic supervision from natural documents. MidTool augments web, PDF, and code with teacher-generated trajectories and also synthesizes trajectories from APIs and MCP skills (Jiang et al., 2026), while GEM extracts workflows from text, synthesizes document-specific tools, and generates complete dialogues with simulated tool responses (Xu et al., 2026). In contrast, we reconstruct records that already contain action–outcome evidence into source-grounded trajectories, then focus on expanding these short episodes into long-horizon supervision with delayed dependencies and multi-task structure.

Agentic training before post-training. GLM-4.5 injects synthetic tool trajectories during long-context mid-training (Zeng et al., 2025); daVinci-Dev builds agent-native software-engineering mid-training data with executable environments (Zeng et al., 2026); Qwen-AgentWorld trains language world models under a continual-pretraining objective (Zuo et al., 2026). These establish that agent behavior can be front-loaded.

Long-context and long-horizon data construction. NExtLong inserts retrieved hard negatives between dependent document spans (Gao et al., 2025), while ACC constructs long-context agentic supervision from observations produced by environment rollouts (Su et al., 2026). In contrast, AnchorStretch identifies an early premise and a later continuation in the same audited trajectory whose decision genuinely depends on that premise, and then increases their separation with complete, topically related agent episodes; ArtifactStitch addresses the complementary problem of constructing coherent multi-task sessions from related source-grounded trajectories.

5 Conclusion

We present Web2LongTraj, a corpus-first alternative to environment-built agent data. It reconstructs source-grounded trajectories from natural web records and scales them with AnchorStretch, which extends within-task dependencies, and ArtifactStitch, which composes related tasks through shared artifacts. Across model scales and both post-training settings, mid-training on these data consistently improves agentic and long-horizon benchmarks while preserving general capabilities. These results show that natural web corpora can provide scalable long-horizon agent supervision without constructing new task environments.

AI USE STATEMENT

We used LLMs to polish wording during manuscript revision and to help locate related work. LLMs also support the data-construction pipeline: Web2LongTraj uses model-based scoring, constrained trajectory reconstruction, observation auditing, and dependency review, and AnchorStretch uses model prompts for split selection and probe construction (Sections 2.1, 2.2 and I). We reviewed all AI-assisted work and take responsibility for the final content, including text, claims, and artifacts produced with the aid of generative AI.

ETHICS STATEMENT

This work uses publicly available web data and existing benchmark suites and does not involve new interaction with human subjects. All materials used in this study were filtered and anonymized to mitigate privacy and safety risks in the source records. Improving long-horizon tool use can make deployed agents more effective but also more harmful. Real-world use should pair them with least-privilege permissions, sandboxing, monitoring, and domain-appropriate human oversight.

REPRODUCIBILITY STATEMENT

The main paper specifies the Web2LongTraj pipeline and both expansion operators (Sections 2.1 to 2.3), and it states the training and evaluation setup used for every comparison (Section 3.1). The appendix records the packing and tokenization details (Section D), the corpus and compilation details (Section A), the evaluation and decoding protocol (Section E), and the exact prompts used in all LLM-mediated stages (Section I). Together these sections document the data transformations, the long-context construction procedure, the benchmark denominators, and the serving protocol needed to reproduce the reported measurements.

REFERENCES

Victor Barres, Honghua Dong, Soham Ray, Xujie Si, and Karthik Narasimhan. τ^2-bench: Evaluating conversational agents in a dual-control environment. *arXiv preprint arXiv:2506.07982*, 2025.

Andrei Z Broder. On the resemblance and containment of documents. In *Proceedings. Compression and Complexity of SEQUENCES 1997 (Cat. No. 97TB100171)*, pp. 21–29. IEEE, 1997.

Shihan Dou, Ming Zhang, Zhangyue Yin, Chenhao Huang, Yujiong Shen, Junzhe Wang, Jiayi Chen, Yuchen Ni, Junjie Ye, Cheng Zhang, et al. Cl-bench: A benchmark for context learning. *arXiv preprint arXiv:2602.03587*, 2026.

Chaochen Gao, Xing Wu, Zijia Lin, Debing Zhang, and Songlin Hu. Nextlong: Toward effective long-context training without long documents. *arXiv preprint arXiv:2501.12766*, 2025.

Dan Hendrycks, Collin Burns, Steven Basart, Andy Zou, Mantas Mazeika, Dawn Song, and Jacob Steinhardt. Measuring massive multitask language understanding. *arXiv preprint arXiv:2009.03300*, 2020.

Fengqing Jiang, Yite Wang, Boyi Liu, Zhaoyang Wang, Canwen Xu, Zhewei Yao, Radha Poovendran, and Yuxiong He. Midtool: Mid-training data synthesis for agentic tool use. *arXiv preprint arXiv:2608.20314*, 2026.

Jiawei Liu, Chunqiu Steven Xia, Yuyao Wang, and Lingming Zhang. Is your code generated by chatgpt really correct? rigorous evaluation of large language models for code generation. *Advances in neural information processing systems*, 36:21558–21572, 2023.

Weize Liu, Yongchi Zhao, Yijia Luo, Mingyu Xu, Jiaheng Liu, Yanan Li, Xiguo Hu, Zhiqi Bai, Yuchi Xu, Bo Zheng, et al. Designer: Design-logic-guided multidisciplinary data synthesis for llm reasoning. In *International Conference on Learning Representations*, volume 2026, pp. 50959–50992, 2026.

Zuxin Liu, Thai Hoang, Jianguo Zhang, Ming Zhu, Tian Lan, Shirley Kokane, Juntao Tan, Weiran Yao, Zhiwei Liu, Yihao Feng, et al. Apigen: Automated pipeline for generating verifiable and diverse function-calling datasets. *Advances in Neural Information Processing Systems*, 37:54463–54482, 2024.

Arindam Mitra, Luciano Del Corro, Guoqing Zheng, Shweti Mahajan, Dany Rouhana, Andres Codas, Yadong Lu, Wei-ge Chen, Olga Vrousgos, Corby Rosset, et al. Agentinstruct: Toward generative teaching with agentic flows. *arXiv preprint arXiv:2407.03502*, 2024.

Team Olmo, Allyson Ettinger, Amanda Bertsch, Bailey Kuehl, David Graham, David Heineman, Dirk Groeneveld, Faeze Brahman, Finbarr Timbers, Hamish Ivison, et al. Olmo 3. *arXiv preprint arXiv:2512.13961*, 2025.

Shishir G Patil, Huanzhi Mao, Fanjia Yan, Charlie Cheng-Jie Ji, Vishnu Suresh, Ion Stoica, and Joseph E Gonzalez. The berkeley function calling leaderboard (bfcl): From tool use to agentic evaluation of large language models. In *Forty-second International Conference on Machine Learning*, 2025.

Guilherme Penedo, Hynek Kydlíček, Anton Lozhkov, Margaret Mitchell, Colin Raffel, Leandro Von Werra, Thomas Wolf, et al. The fineweb datasets: Decanting the web for the finest text data at scale. *Advances in Neural Information Processing Systems*, 37:30811–30849, 2024.

Bowen Peng, Jeffrey Quesnelle, Honglu Fan, and Enrico Shippole. Yarn: Efficient context window extension of large language models. In *International Conference on Learning Representations*, volume 2024, pp. 31932–31951, 2024.

Akshara Prabhakar, Zuxin Liu, Ming Zhu, Jianguo Zhang, Tulika Manoj Awalgaonkar, Shiyu Wang, Zhiwei Liu, Haolin Chen, Thai Hoang, Juan Carlos Niebles, et al. Apigen-mt: Agentic pipeline for multi-turn data generation via simulated agent-human interplay. *Advances in Neural Information Processing Systems*, 38, 2026.

Yujia Qin, Shihao Liang, Yining Ye, Kunlun Zhu, Lan Yan, Yaxi Lu, Yankai Lin, Xin Cong, Xiangru Tang, Bill Qian, et al. Toolllm: Facilitating large language models to master 16000+ real-world apis. In *International Conference on Learning Representations*, volume 2024, pp. 9695–9717, 2024.

Negin Raoof, Richard Zhuang, Marianna Nezhurina, Etash Guha, Atula Tejaswi, Ryan Marten, Charlie F Ruan, Tyler Griggs, Alexander Glenn Shaw, Hritik Bansal, et al. Openthoughts-agent: Data recipes for agentic models. *arXiv preprint arXiv:2606.24855*, 2026.

David Rein, Betty Li Hou, Asa Cooper Stickland, Jackson Petty, Richard Yuanzhe Pang, Julien Dirani, Julian Michael, and Samuel R Bowman. Gpqa: A graduate-level google-proof q&a benchmark. *arXiv preprint arXiv:2311.12022*, 2023.

Qisheng Su, Zhen Fang, Shiting Huang, Yu Zeng, Yiming Zhao, Kou Shi, Ziao Zhang, Lin Chen, Zehui Chen, Lijun Wu, et al. Acc: Compiling agent trajectories for long-context training. *arXiv preprint arXiv:2605.21850*, 2026.

Mirac Suzgun, Nathan Scales, Nathanael Schärli, Sebastian Gehrmann, Yi Tay, Hyung Won Chung, Aakanksha Chowdhery, Quoc Le, Ed H Chi, Denny Zhou, et al. Challenging big-bench tasks and whether chain-of-thought can solve them. In *Findings of the Association for Computational Linguistics: ACL 2023*, pp. 13003–13051, 2023.

Yubo Wang, Xueguang Ma, Ge Zhang, Yuansheng Ni, Abhranil Chandra, Shiguang Guo, Weiming Ren, Aaran Arulraj, Xuan He, Ziyan Jiang, et al. Mmlu-pro: A more robust and challenging multi-task language understanding benchmark. *Advances in Neural Information Processing Systems*, 37:95266–95290, 2024.

Di Wu, Hongwei Wang, Wenhao Yu, Yuwei Zhang, Kai-Wei Chang, and Dong Yu. Longmemeval: Benchmarking chat assistants on long-term interactive memory. *arXiv preprint arXiv:2410.10813*, 2024.

Yiheng Xu, Dunjie Lu, Zhennan Shen, Junli Wang, Zekun Wang, Yuchen Mao, Caiming Xiong, and Tao Yu. Agenttrek: Agent trajectory synthesis via guiding replay with web tutorials. In *International Conference on Learning Representations*, volume 2025, pp. 79822–79843, 2025a.

Zhangchen Xu, Adriana Meza Soria, Shawn Tan, Anurag Roy, Ashish Sunil Agrawal, Radha Poovendran, and Rameswar Panda. Toucan: Synthesizing 1.5 m tool-agentic data from real-world mcp environments. *arXiv preprint arXiv:2510.01179*, 2025b.

Zhihao Xu, Rumei Li, Jiahuan Li, Rongxiang Weng, Jingang Wang, Xunliang Cai, and Xiting Wang. Unlocking implicit experience: Synthesizing tool-use trajectories from text. *arXiv preprint arXiv:2601.10355*, 3, 2026.

John Yang, Akshara Prabhakar, Karthik Narasimhan, and Shunyu Yao. Intercode: Standardizing and benchmarking interactive coding with execution feedback. *Advances in Neural Information Processing Systems*, 36: 23826–23854, 2023.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. React: Synergizing reasoning and acting in language models. *arXiv preprint arXiv:2210.03629*, 2022.

Fan Yin, Zifeng Wang, I-Hung Hsu, Jun Yan, Ke Jiang, Yanfei Chen, Jindong Gu, Long Le, Kai-Wei Chang, Chen-Yu Lee, et al. Magnet: Multi-turn tool-use data synthesis and distillation via graph translation. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 32600–32616, 2025.

Aohan Zeng, Xin Lv, Qinkai Zheng, Zhenyu Hou, Bin Chen, Chengxing Xie, Cunxiang Wang, Da Yin, Hao Zeng, Jiajie Zhang, et al. Glm-4.5: Agentic, reasoning, and coding (arc) foundation models. *arXiv preprint arXiv:2508.06471*, 2025.

Ji Zeng, Dayuan Fu, Tiantian Mi, Yumin Zhuang, Yaxing Huang, Xuefeng Li, Lyumanshan Ye, Muhang Xie, Qishuo Hua, Zhen Huang, et al. davinci-dev: Agent-native mid-training for software engineering. *arXiv preprint arXiv:2601.18418*, 2026.

Yanzhao Zhang, Mingxin Li, Dingkun Long, Xin Zhang, Huan Lin, Baosong Yang, Pengjun Xie, An Yang, Dayiheng Liu, Junyang Lin, et al. Qwen3 embedding: Advancing text embedding and reranking through foundation models. *arXiv preprint arXiv:2506.05176*, 2025.

Yujie Zhao, Boqin Yuan, Junbo Huang, Haocheng Yuan, Zhongming Yu, Haozhou Xu, Lanxiang Hu, Abhilash Shankarampeta, Zimeng Huang, Wentao Ni, et al. Ama-bench: Evaluating long-horizon memory for agentic applications. *arXiv preprint arXiv:2602.22769*, 2026.

Jeffrey Zhou, Tianjian Lu, Swaroop Mishra, Siddhartha Brahma, Sujoy Basu, Yi Luan, Denny Zhou, and Le Hou. Instruction-following evaluation for large language models. *arXiv preprint arXiv:2311.07911*, 2023.

Terry Yue Zhuo, Minh Chien Vu, Jenny Chim, Han Hu, Wenhao Yu, Ratnadira Widyasari, Imam Nur Bani Yusuf, Haolan Zhan, Junda He, Indraneil Paul, et al. Bigcodebench: Benchmarking code generation with diverse function calls and complex instructions. In *International Conference on Learning Representations*, volume 2025, pp. 66602–66656, 2025.

Yuxin Zuo, Zikai Xiao, Li Sheng, Fei Huang, Jianhong Tu, Yuxuan Liu, Tianyi Tang, Xiaomeng Hu, Yang Su, Qingfeng Lan, et al. Qwen-agentworld: Language world models for general agents. *arXiv preprint arXiv:2606.24597*, 2026.

A Corpus and Compilation Details

Part of the pool we score had already passed a round of data-quality pre-filtering. Agentic scores across that same set still spread over the full range from 0 to 5. This is the clearest evidence we have that agentic value is a separate signal: a document can be authoritative, well written and reusable, and still teach nothing about how to act. The dimension means on that set make the point concrete. Procedurality averages 2.78 and knowledge quality 3.93, while observability averages 1.65 and decision density 1.55. The text is reliable and often lists steps; what it does not contain is the observe-and-decide structure that agent training needs, which is why records of that kind feed the rubric but not the compiler.

Compilation yield follows the same logic. Under a prompt that requires every observation to be literal machine output, records built around code and error messages yield several times more than discussion or purely factual answers, because only the former quote what a program returned. The prompt matters as much as the source: a prompt that transcribes the record word for word, instead of reconstructing the episode, produces trajectories 2.4 times shorter whose observations are usually a human sentence such as "it worked".

Readiness-score ceilings. The structural ceiling is 1 for unreadable or unsafe text, text that is mostly advertising or metadata, and breaking news; 2 for encyclopedic statements and paper abstracts without method; 3 for patent or standard abstracts that list only scope or claims and for API references that list only parameters; and 5, that is no ceiling, otherwise.

Compilation constraints. Tool schemas are inferred from observed calls, and a parameter is marked required only when every call includes it. Every concrete outcome must be supported by the source; social replies and unsupported success claims cannot become observations; each trajectory must contain at least two actions and two observations with well-formed tags; and any violation causes rejection rather than repair.

Scoring and auditing at production scale. On the production run the final Agentic Readiness Score correlates with the dimension mean at 0.911 and rule violations occur on at most 0.03% of records, so the ceiling fires on structural defects rather than on ordinary variation. In the observation audit, provenance is measured but is not an admission threshold: we record character n-gram coverage of every observation against its source and export it for stratified analysis, because a source-conditioned reconstruction of a real command output is acceptable whereas a verbatim human sentence is not. An invented confirmation such as "Measure 'X' created successfully" is indistinguishable from real output by surface pattern, while genuine `polars` tables and Gradle logs are easily mistaken for prose by a pattern matcher. A deterministic rule baseline is retained as a reproducible control and agrees with the judge on 80.04% of observations.

AnchorStretch admission checks. Every admitted split must pass four mechanical checks: $p \parallel s$ must exactly reconstruct the original body, the anchor must occur exactly once without leaking into distractors or the resume cue, the anchor must appear within the first 75% of the source, and s must form a complete reasoning and action unit.

AnchorStretch controls. For each (main, variant, length) tuple, the three retrieval policies share the main, the anchor, the continuation, the resume cue, the tool header, the token budget and the random seed, so that the distractor policy is the only difference between them. Where lexical retrieval finds no eligible neighbour, the lexical variant uses the exact random distractors, so the treatment is applied only where it is supported; about 43% of rows are byte-identical to their random counterpart for this reason. Resume cues are randomized over four strengths, explicit, referential, minimal and none, at 25% each, so that no single phrase becomes a retrieval trigger. Lexical retrieval represents each trajectory by its task signature: the first user request, truncated to 1,800 characters, together with the names of up to 24 tools the trajectory calls. The signature is tokenized into lowercase words, character bigrams for CJK text, and tool-name tokens; answers and observations never enter it. A distractor is additionally excluded when it duplicates the main record, repeats a body or signature hash, or contains the anchor.

Table 4: AnchorStretch production, per retrieval variant. Dependency distance is the token distance from the anchor to the continuation. Rows are identical across the three variants by construction; after exact tokenization we remove the union of overlength rows across variants so that they stay row-aligned, and never truncate a retained trajectory.

Window	Rows	Tokens/row	Dependency distance	Distractors/row	Tokens
64K	68,494	53,540	51,944	32.8	3.67B
128K	28,433	110,675	109,059	68.8	3.15B
256K	16,079	228,450	226,826	143.0	3.67B
Total	113,006				10.5B

B Dependency Admission Ablation

Each AnchorStretch split comes from one of the two branches of Section 2.2. The *existing-dependency* branch extracts an anchor and continuation already present in the trajectory; the *planted-probe* branch appends a follow-up task that requires an earlier fact. Both branches always pass the same mechanical gates. Table 5 varies only the final admission rule: *Self-scored* trusts the construction model's score, whereas *Reviewed* uses a separate review call that does not see that score. To avoid compensating for stricter review by repeating fewer surviving trajectories, which would make the comparison unfair, every configuration is capped at 1B AnchorStretch tokens in each Stage 3 sub-stage.

Table 5: Dependency-admission ablation. Each branch uses either the construction model's own score (Self-scored) or an independent reviewer (Reviewed). All rows use SeedTraj in Stage 2, hybrid retrieval, and 1B AnchorStretch tokens per Stage 3 sub-stage.

Existing dependency	Planted probe	BFCL	IC-Bash	τ^2-bench	LongMemEval	AMA	CL	MBPP+	BCB
Instruct SFT									
Self-scored	Self-scored	26.6	77.1	10.6	25.2	27.5	22.7	47.6	25.1
Reviewed	Self-scored	29.5	76.9	11.3	26.0	26.4	21.9	52.1	26.5
Self-scored	Reviewed	26.8	**79.3**	10.8	26.0	26.6	**23.1**	51.1	**28.3**
Reviewed	Reviewed	**29.6**	78.7	**12.2**	**27.0**	**27.8**	**23.1**	**53.4**	26.8
Think SFT									
Self-scored	Self-scored	22.7	66.6	18.6	23.2	23.2	15.6	36.0	23.1
Reviewed	Self-scored	23.8	68.3	20.8	25.2	20.4	15.0	34.7	23.2
Self-scored	Reviewed	**24.1**	70.8	19.1	25.0	20.6	14.9	31.0	20.5
Reviewed	Reviewed	23.9	**76.5**	**21.0**	**29.0**	**24.1**	**15.9**	**36.8**	**23.6**

Independent review produces better supervision. Reviewing both existing dependencies and planted probes improves every benchmark over self-scoring alone under both SFT settings. This confirms that dependency verification materially improves the training signal.

C Graph Construction

Artifacts and environment tags. Artifacts are file names with their extensions, matched by basename. Stack, operating system, and workflow phase are assigned by conservative keyword rules, and a tag stays unknown when no rule fires. The stack (13 categories, such as Python, Java, Node.js, and shell) and the operating system (Windows, macOS, or Linux) are read from the trajectory body; the phase is read from the first user request and ordered as setup, configure, implement, debug, test, deploy, and operate. A directed candidate edge runs from a trajectory that creates or modifies an artifact to one that reads or modifies it. The edge requires the same stack and no operating-system conflict, with unknown tags compatible with any value, and the phase of its target may precede that of its source by at most one step.

Edge corroboration. Let $N_k(v)$ be the $k = 30$ nearest neighbors of node v under Qwen3-Embedding-8B (Zhang et al., 2025), and define $c(u, v) = \mathbf{1}[v \in N_k(u) \vee u \in N_k(v)]$, so that a pair counts as neighbors when either node is among the other's k nearest neighbors. For an artifact

a inducing candidate edge set E_a, the per-artifact hit rate and its enrichment over chance are

$$h(a) = \frac{1}{|E_a|} \sum_{(u,v)\in E_a} c(u,v), \qquad \lambda(a) = \frac{h(a)}{2k/N}, \tag{4}$$

where $N = 1{,}154{,}483$ is the pool size. The neighbor relation contains at most kN of the $N(N-1)/2$ unordered pairs, so under independence between artifact sharing and embedding geometry a random pair satisfies c with probability at most $2k/N \approx 5.2 \times 10^{-5}$, and $\lambda(a)$ is therefore a conservative estimate of the enrichment. We retain an artifact when $h(a) \geq 0.10$ and $|E_a| \geq m$ with $m = 5$. Measured hit rates on retained edge sets fall between 12.7% and 15.3%, hence λ lies between about 2,400 and 2,900. Artifact sharing on its own is a weak and partly spurious signal; Equation (4) is what separates a genuinely shared project file from a common placeholder such as `config.json`. The resulting graph yields 13,164 multi-task sessions containing 107.9M tokens, with 5.1 tasks per session on average.

The session declaration. We merge per-trajectory schemas by taking the union of parameter names and the intersection of required parameter names, so that the calls already present in the session remain well-formed under the session header.

D Training Details

Models and budgets. Olmo 3 releases its data, code, intermediate checkpoints, and long-context recipe (Olmo et al., 2025). Most controlled experiments begin from the Olmo-3-7B Stage 1 checkpoint, which is the checkpoint after pretraining. We then reproduce Stage 2 and Stage 3 while changing only the training data specified by each comparison. Mid-training uses next-token prediction over all tokens. Stage 2 contains 40B tokens at sequence length 8,192, including 1.90B SeedTraj tokens. Stage 3 contains 20B tokens, split evenly between long-context data and official Olmo 3 short-context data. Its 64K, 128K, and 256K sub-stages use YaRN factors 8, 16, and 32 relative to the 8,192-token pretraining window (Peng et al., 2024). Because the released Olmo 3 models were trained only up to 64K contexts, we sample Dolma 3 Longmino documents and pack token-matched 64K, 128K, and 256K controls; throughout the paper, these controls are called Olmo 3 long data. In fixed-budget comparisons, our data replaces equal-sized subsets of Olmo 3 long data matched to its document-length distribution, preserving the token count of every sub-stage. We also add our data to the full 150B-token Olmo-3-7B recipe and evaluate our synthesized data at a larger scale with Olmo-3-32B.

Post-training mixtures. Every mid-trained checkpoint is evaluated after both Instruct and Think SFT. Within each setting, all configurations use the same data, training budget, and recipe. The Instruct setting uses Olmo 3's released Instruct mixture (Olmo et al., 2025), which contains 1.92B tokens. The released Think mixture contains no agent trajectories, so we construct a standardized Think SFT mixture that preserves tool-use behavior learned during mid-training. By unique trainable tokens, it contains 57.9% Dolci-Think (Olmo et al., 2025) and 32.1% DLR-Web (Liu et al., 2026) for general reasoning, and 7.5% Nemotron-SFT-Agentic, 2.4% OpenThoughts-Agent (Raoof et al., 2026), and 0.1% xLAM function calling (Liu et al., 2024) for agentic reasoning. Thus, agentic data accounts for 10.0% of the mixture. The 90/10 balance maintains broad reasoning coverage while supplying multi-turn tool-use examples. The mixture contains 2.98B tokens from 771,177 documents, of which 2.36B are trainable. We render it with the standard `<think>`, `<tool_call>`, and `<tool_response>` protocol. Unlike mid-training, SFT masks the loss on system, user, and tool-response spans, which are inputs supplied by the environment.

Tokenization and row alignment. Rows are aligned across retrieval variants in a fixed order that cannot be permuted. First the three variants are intersected so that every row has a counterpart in each. Then rows whose exact tokenization exceeds the block size minus one are removed as the union across variants, never truncated, since truncation would remove the continuation and with it the property being measured. Removal rates are 0.261% at 64K, 0.232% at 128K and 0.100% at 256K. The random variant never overflows because its distractors are sampled uniformly in length, whereas lexical-retrieval distractors are longer.

Long-context block packing. Each block is one tokenized trajectory, one end-of-sequence marker, and filler to the block boundary, with the trajectory aligned to token 0. Intra-document masking uses the end-of-sequence marker as the document boundary, so filler never attends to the body. Only the system header is re-rendered at training time; the body is byte-identical to the delivered data. We add no chat special tokens that the base model has not seen during pretraining. Per-variant long-context totals are 3.5B tokens at 64K, 3B at 128K, and 3.5B at 256K, identical across variants. Filler is drawn from a corpus region never seen in Stage 2 and is byte-identical across variants at the same block index, so it cannot introduce a difference between them.

ArtifactStitch placement. ArtifactStitch sessions are pre-packed at 256K tokens and enter the 256K sub-stage as 425 blocks totaling 111.4M tokens, placed first in the data order. The dataset concatenates arrays before cutting windows, so a pre-packed block survives intact only when its array starts on a window boundary; placing this array first aligns both it and the long-context array that follows. The Olmo 3 long data ballast for that sub-stage is shortened by the same number of tokens, so the budget and step count are unchanged and the configuration without ArtifactStitch is an ordered subset of the same Olmo 3 long data tokens.

E Evaluation Protocol

Agentic benchmarks. BFCL-V4 (Patil et al., 2025) evaluates tool use over all 22 official categories, including multi-turn and agentic tasks and the `irrelevance` item, which rewards declining when no declared tool applies. LongMemEval (Wu et al., 2024) uses the full 500-question `longmemeval_s` set in full-history mode. AMA-Bench (Zhao et al., 2026) comprises 208 episodes and 2,496 question-answer pairs over long agent traces. CL-bench (Dou et al., 2026) evaluates rubric-based long-context task completion over 1,899 questions. We serve these four benchmarks with a 256K context window, and Qwen3-32B grades the three that require an external judge.

InterCode-Bash (Yang et al., 2023) places the model in a Docker-backed shell, where it issues Bash commands and observes execution feedback until it completes a requested file-system operation. We evaluate the complete 200-task NL2Bash adaptation over its four official file-system environments and report mean environment reward. τ^2-bench (Barres et al., 2025) evaluates conversational agents in dual-control settings where both the agent and a simulated user can call tools to modify a shared environment. We use the complete base splits of airline, retail, and telecom, comprising 50, 114, and 114 tasks, respectively. The user simulator and auxiliary LLM roles use Qwen3-32B.

Code benchmarks. Because the mined trajectories mainly use shell, file, and code-execution tools, we measure coding transfer on MBPP+ (Liu et al., 2023) and BigCodeBench (Zhuo et al., 2025). MBPP+ contains 378 function-generation problems. We use the instruction setting of BigCodeBench, which contains 1,140 natural-language tasks that require composing calls across multiple libraries. Across all tables, BFCL, IC-Bash, AMA, CL, and BCB denote BFCL-V4, InterCode-Bash, AMA-Bench, CL-bench, and BigCodeBench, respectively; τ^2-bench, LongMemEval, and MBPP+ are shown without abbreviation.

Metrics. Except for IC-Bash, all scores in this paper are percentages: τ^2-bench reports pass^1 (task success rate), MBPP+ and BCB report pass@1, and the remaining benchmarks report accuracy. IC-Bash reports the mean environment reward, a partial-credit score in $[0, 1]$ multiplied by 100.

Decoding and comparison. We evaluate every benchmark on its complete official set using greedy decoding at temperature 0. Greedy decoding is the prescribed setting for LongMemEval, AMA-Bench, τ^2-bench, and the two code benchmarks. BFCL-V4 and InterCode-Bash default to temperatures of 0.001 and 0.01, respectively, which are both near-greedy. Deterministic decoding removes sampling variance from the comparison, so each configuration needs to be evaluated only once.

Judged benchmarks use Qwen3-32B as a fixed open judge at temperature 0. Using each benchmark's prescribed judge would introduce grader variation across benchmark columns, so we hold the judge fixed and validate this choice. We score the same outputs with the prescribed proprietary judges and use Claude Opus 5 to adjudicate every disagreement. Qwen3-32B achieves essentially the same accuracy as GPT-4o and GPT-5.1 on these tasks. We therefore use the open judge throughout, keeping the evaluation reproducible without proprietary access.

F Additive Olmo-3-7B Results

Table 6: Additive Olmo-3-7B comparison: 12B tokens from the Web2LongTraj corpus are appended to the 150B-token Olmo 3 long data recipe without removing control data.

Data Recipe	BFCL	IC-Bash	τ^2-bench	LongMemEval	AMA	CL	MBPP+	BCB
Instruct SFT								
Olmo 3 long data (150B)	27.1	77.0	12.1	25.6	26.3	23.4	50.8	26.1
Olmo 3 long data (150B) + Web2LongTraj (12B)	**29.1**	**80.3**	**12.4**	**28.0**	**30.5**	**24.2**	**53.7**	**32.6**
Think SFT								
Olmo 3 long data (150B)	23.6	75.0	19.6	26.6	12.2	16.0	40.2	16.1
Olmo 3 long data (150B) + Web2LongTraj (12B)	**24.8**	**77.1**	**22.3**	**33.3**	**17.3**	**16.1**	**41.5**	**20.9**

The same conclusion holds in the additive setting (Table 6): appending 12B tokens from our corpus to the complete 150B-token Olmo 3 long data recipe improves all eight benchmarks under both SFT settings. The benefit therefore does not rely on replacing control data; our corpus can also be added on top of it to further improve performance.

G Context-Length Scaling Results

Table 7 reports all eight benchmarks behind Figure 5. At every maximum window and under both SFT settings, SeedTraj + AnchorStretch outperforms the token-matched SeedTraj control on every benchmark.

Table 7: Context-length scaling for AnchorStretch. Each maximum window includes all preceding Stage 3 sub-stages. The SeedTraj and SeedTraj + AnchorStretch recipes are token-matched within each pair; budgets differ across pairs.

Max context	Data Recipe	BFCL	IC-Bash	τ^2-bench	LongMemEval	AMA	CL	MBPP+	BCB
Instruct SFT									
64K	SeedTraj	25.5	76.3	10.1	14.2	24.8	20.4	40.2	25.3
64K	SeedTraj + AnchorStretch	**28.0**	**78.1**	**10.3**	**16.0**	**26.4**	**23.0**	**50.0**	**27.5**
128K	SeedTraj	26.5	77.8	10.9	16.2	24.7	20.7	50.5	26.8
128K	SeedTraj + AnchorStretch	**28.4**	**78.6**	**11.2**	**22.0**	**26.4**	**23.3**	**51.1**	**27.0**
256K	SeedTraj	25.4	77.6	10.8	28.2	25.2	22.0	47.1	25.4
256K	SeedTraj + AnchorStretch	**29.1**	**80.2**	**11.0**	**29.9**	**27.0**	**23.7**	**50.8**	**26.8**
Think SFT									
64K	SeedTraj	21.5	56.7	20.9	11.2	14.6	14.7	32.0	21.5
64K	SeedTraj + AnchorStretch	**24.4**	**60.4**	**21.8**	**13.6**	**15.3**	**15.5**	**35.4**	**22.5**
128K	SeedTraj	20.8	64.1	18.0	15.4	18.8	15.4	23.5	22.5
128K	SeedTraj + AnchorStretch	**22.7**	**66.4**	**22.6**	**19.4**	**21.0**	**16.2**	**31.2**	**24.0**
256K	SeedTraj	23.0	68.9	19.7	26.5	16.6	15.8	33.3	21.2
256K	SeedTraj + AnchorStretch	**24.3**	**70.0**	**19.8**	**32.6**	**20.9**	**16.3**	**36.0**	**21.3**

H Full General-Capability Results

As Table 8 shows, our data preserves general capability. Under Think SFT, the full Web2LongTraj recipe improves all five benchmarks over Olmo 3 long data, including $+9.3$ on BBH and $+3.5$ on GPQA-Diamond. Under Instruct SFT, it improves MMLU-Pro, GPQA-Diamond, and BBH and stays within one point of Olmo 3 long data on MMLU and IFEval. Replacing matched subsets of Olmo 3 long data with our data preserves knowledge and instruction following while improving multi-step reasoning.

I Prompts

Every prompt below is reproduced verbatim from the production scripts, in the exact form used to build the delivered corpora, with five non-ASCII characters rewritten so that the source compiles

Table 8: Full general-capability results for fixed-budget Olmo-3-7B recipes; each SFT block uses the same checkpoint, 60B-token budget, and post-training recipe.

Data Recipe	MMLU	MMLU-Pro	GPQA-Diamond	BBH	IFEval
Instruct SFT					
Olmo 3 long data	**59.7**	31.7	30.3	41.6	80.4
SeedTraj	59.4	31.4	29.3	42.1	80.4
SeedTraj + AnchorStretch	59.1	31.5	**33.3**	43.6	**81.0**
SeedTraj + ArtifactStitch	59.0	**32.5**	28.3	42.2	79.1
Web2LongTraj	58.8	32.1	30.8	**43.8**	80.0
Think SFT					
Olmo 3 long data	68.5	46.3	26.8	54.2	64.7
SeedTraj	68.7	46.9	25.3	61.3	61.9
SeedTraj + AnchorStretch	68.8	45.5	**30.8**	59.0	**67.5**
SeedTraj + ArtifactStitch	68.4	**47.2**	24.7	57.2	64.3
Web2LongTraj	**69.0**	47.0	30.3	**63.5**	67.3

under pdfLaTeX: one emoji, one arrow, one em dash and one pair of curly quotes. Each is issued as a system message; the user message carries the record, the trajectory or the observation being judged, in the shape the surrounding text describes. ArtifactStitch has no entry here because it calls no language model: its graph is built from artifact strings and an embedding index, its walks are deterministic, and its bridge turns are drawn from a fixed template, so nothing in a session is generated.

I.1 Agentic Readiness Scoring

The rubric of Section 2.1, gate one. The ceiling rules appear here as the hard gates that cap ARS before the dimension scores are aggregated.

Prompt for agentic readiness scoring

```
You are a data-selection expert for LLM mid-training. Your job is to judge whether the
given text can serve as effective mid-training data for improving a model's agentic and
tool-using abilities, including tool use, SOP following, selecting the right tool/file/
reference, multi-step execution, and observing results and self-correcting.

The user message is one data record in JSON, with fields uuid, title, url, language,
domain_l2, text. Score based only on this record.

Note: you are NOT judging general knowledge quality. Being authoritative, fluent, entity-
rich, or having a high knowledge score does NOT mean a high score. Pure encyclopedia
entries, pure facts, paper abstracts, news, marketing copy, patent legal status, and book
 metadata, even if their knowledge quality is fine, should still get a low score if they
contain no procedure, action, condition, verification, or structure convertible into an
agent trajectory.

Output only one valid JSON object. Do not output any explanation, do not output Markdown,
 do not output your thinking.

Core judging goal:
A high-scoring text should help the model learn at least one of:
1. Completing a task step by step.
2. Choosing among multiple methods, tools, workflows, diagnostic paths, or configurations.

3. Deciding the next step based on conditional branches, errors, logs, or observed
results.
4. Following an SOP, tutorial, manual, API doc, README Usage, runbook, notebook, standard
 procedure, or clinical/engineering/experimental guideline.
5. Being rewritten into an operating procedure, reference doc, bridge-recast data, or
agent trajectory.

Low-scoring texts are usually:
Pure facts, pure encyclopedia definitions, ordinary news, subjective opinions, marketing
soft-ads, paper titles or abstracts, book metadata, patent legal status, low-density
tables, text that only names a method without explaining how to do it, pure code without
explanation, pure logs without a fix path, OCR noise, or page templates.

First apply the hard gates:
```

```
If the text is unreadable, severely garbled, missing its main body, severely truncated,
mostly ads/SEO/metadata, or contains clearly dangerous/illegal operations, final_score
must be 0 or 1.
If it is a pure factual encyclopedia or textbook concept explanation but has no steps,
actions, conditions, or verification, final_score is at most 2.
If it is a paper abstract with no method details, final_score is at most 2.
If it is a patent/standard/regulation abstract that only lists scope or claims without
operational steps, final_score is at most 3.
If it is an API reference with only parameter enumeration and no usage flow or example,
final_score is at most 3.
If it is pure breaking news with no lasting knowledge, final_score is at most 1.
If the text has complete steps but is high time-sensitivity or high-risk (e.g., medical,
legal, tax, finance, engineering safety), it may get a higher score, but
high_risk_or_time_sensitive must be true.

Scoring dimensions, each 0-5:
procedurality: whether it has clear steps, workflow, SOP, tutorial, experimental method,
or operating procedure.
actionability: whether actions are concrete and executable by a human or an agent,
including commands, parameters, configs, material ratios, APIs, paths, forms, operation
targets.
observability: whether it has expected outputs, error messages, logs, tests, acceptance
criteria, observed results, metrics, or verification methods.
decision_density: whether it has conditional branches, method selection, applicable/non-
applicable scope, troubleshooting, fallback, or error recovery.
trajectory_convertibility: whether it can be turned, without fabricating facts, into a
runbook, reference, bridge data, or agent trajectory.
knowledge_quality: whether the content is objective, reliable, reusable, low-noise, non-
marketing, non-purely-subjective, and free of obvious hallucination.

Final final_score 0-5:
0 = Reject. Unreadable, unsafe, obvious garbage, severely wrong, no training value.
1 = Low value. Almost no agentic signal; mostly news, facts, metadata, marketing, or
subjective content.
2 = Weak candidate. Has task or method signal, but the process is incomplete, non-
executable, or hard to turn into a trajectory.
3 = Usable natural corpus. Has some steps, methods, conditions, or decisions; can enter
raw mid-training at low/medium weight.
4 = Strong procedure-ready corpus. Clear SOP/tutorial/manual/guideline/runbook, concrete
actions, with constraints or conditions; suitable for procedure extraction.
5 = Excellent trajectory-ready corpus. Beyond a clear process, it also has observation,
verification, logs, error-fix, test output, notebook output, or strong troubleshooting
paths; suitable for bridge recast or trajectory synthesis.

recommended_use, choose exactly one:
discard: discard
low_weight_raw: keep at low weight
raw_midtraining: use as natural process corpus for training
procedure_extraction: suitable to extract into an SOP/runbook
trajectory_synthesis: suitable as a bridge-recast or agent-trajectory seed
reference_only: suitable only as a standard/spec/table/background reference
human_review: high-risk or high time-sensitivity, needs human/domain review

Output JSON in exactly the following format; all fields are required and scores must be
integers from 0 to 5:
{
  "is_readable": true,
  "is_mostly_metadata": false,
  "high_risk_or_time_sensitive": false,
  "content_type": "explicit_sop/implicit_procedure/troubleshooting/tutorial/manual/
standard_or_regulation/notebook_or_code_walkthrough/qa_solution/patent_or_spec/
paper_or_report/pure_fact/news_or_narrative/marketing/metadata_or_bibliography/data_table
/noisy_other",
  "scores": {
    "procedurality": 0,
    "actionability": 0,
    "observability": 0,
    "decision_density": 0,
    "trajectory_convertibility": 0,
    "knowledge_quality": 0
  },
  "final_score": 0,
  "recommended_use": "discard/low_weight_raw/raw_midtraining/procedure_extraction/
trajectory_synthesis/reference_only/human_review",
  "reason": "a concise reason in English, no more than 40 words"
}

Now evaluate the data in the user message and output only the JSON.
```

I.2 CONSTRAINED COMPILATION

Gate two. The four constraints of Section 2.1 are the rules on observations, on rejection, and on the minimum of two action and observation cycles. The instruction that an observation must be the literal output a tool produced, and never a paraphrase of it, is what the audit of the next subsection then verifies independently.

Prompt for constrained compilation

```
You are building high-quality agent-trajectory training data. You are given ONE real text
 that documents how a problem was actually worked on and solved: a forum/Q&A thread, a
GitHub issue or PR, a notebook, a shell/terminal session, a troubleshooting log, a
tutorial, etc.

Reconstruct it into a realistic, COMPLETE ReAct-style agent trajectory -- what it would
look like if a capable agent had actually performed this task in a real environment.
RECONSTRUCT, do not transcribe: keep every real FACT from the source (the real commands,
the real values, the real result of each step, the final resolution), but render them the
 way a real agent run actually looks -- genuine step-by-step reasoning, real tool calls,
and LITERAL tool outputs. This reshaping is expected. What is NOT allowed is inventing
facts that are not in, or implied by, the source (a new result, new numbers, a different
outcome, or steps that never happened).

Only produce a trajectory IF the source contains a REAL, multi-step solving process (
actions that were actually tried and their real outcomes). If it is pure discussion/
opinion, a prescriptive how-to / SOP with no real execution, or advice-only with no real
results, REJECT it (see WHEN TO REJECT).

Return ONLY one JSON object, no markdown, no commentary:
{"recast_ok": true, "reject_reason": "", "task": "<one-line task, from the source>", "
steps": [ <step>, <step>, ... ]}
Each <step> is exactly one of:
{"t": "think", "text": "<the agent's internal reasoning, in the SOURCE language>"}
{"t": "say",   "text": "<a natural user-facing message the agent says to the user, in the
 SOURCE language>"}
{"t": "action", "name": "<short snake_case tool name>", "arguments": { <real parameters>
}}
{"t": "observation", "text": "<the LITERAL result the tool/environment returned>"}
{"t": "final", "text": "<the final answer to the user>"}
Output STRICT JSON only -- no comments, no trailing commas; escape newlines and quotes
inside string values (important for code/logs).

HOW EACH FIELD MUST LOOK -- this is the whole point:

1. think -- REAL, substantial reasoning, NOT one line. Write the actual chain of thought
a competent agent would have: what it already knows, its hypothesis, why it chooses this
next action, and -- after each observation -- how it INTERPRETS that result and decides
what to do next. Expand the source's terse reasoning into genuine agent CoT, but add NO
new external facts. Several sentences per think is normal and expected.

2. say -- natural language spoken TO THE USER, rendered WITHOUT tags. Use it like a real
agent does: an opening line stating the plan, short explanations between phases, and a
brief closing summary. Do NOT push everything into think -- a real trajectory also talks
to the user. Include at least one say.

3. action -- a real tool call with the real command / code / parameters from the source.
Use a precise snake_case name: prefer run, execute_code, read_file, write_file, edit_file,
 search_web, browse, query_db, call_api, inspect, install, configure, test; if none fits
use a specific one (e.g. git_rebase, kubectl_apply). "arguments" is a JSON object of the
concrete values, e.g. {"command": "cargo build --features bindgen"}, {"path": "settings.
py"}. Do NOT force everything into "run", and do NOT invent commands the actor never ran.

4. observation -- THE SINGLE MOST IMPORTANT RULE. An "observation" is the LITERAL output
the tool/environment produced: an error message WITH its code, a stack trace, stdout/
stderr, a log line, a return value, file contents, a query result, an API JSON response.
It must read like real machine output, NEVER like a human sentence.
   - If the source SHOWS the real output -> copy it verbatim (trim long parts with "...").

   - If the source only DESCRIBES the outcome (e.g. "didn't work", "it worked", "same
error", "now it's fast") -> RECONSTRUCT the actual literal output THIS specific command
would really produce, consistent with the source's outcome: same success/failure, same
kind of error, and the exact values / paths / names the source does give. Make it look
like the real tool output.
   - FORBIDDEN as an observation (these are the exact bad cases to eliminate): a
paraphrase or one-sentence summary of the result, an opinion, thanks, or any social
remark. NEVER write things like "didn't work", "it worked now", "Works a charm thank you :
```

```
D", "they seemed to have the correct values", "The event fires but clicking does nothing",
 "(no output)", "[listed the packages]". Each of those MUST instead be the real thing:
the actual error text, the actual printed values, the actual listing.
   - Do NOT invent facts to fill an observation: keep the real outcome and the real
values the source gives; reconstruct only the SURFACE FORM (how the output literally
looks). If you cannot even tell the outcome (success or failure) of a step from the
source, that step has NO real observation -- drop the step.

5. final -- one faithful closing answer that matches the source's real resolution.

HARD RULES -- break any one and set recast_ok=false:
- Keep the source language (Chinese stays Chinese).
- Every action is a real step the actor performed; every observation is that step's real
result (verbatim, or faithfully reconstructed as in rule 4). No invented steps, no
invented outcomes, and NEVER dress up someone's advice / opinion / question as an
observation.
- Strip site-specific interface and author metadata -- keep only substantive content.
- A real trajectory has at least two genuine action->observation cycles, real reasoning
between them, and reaches (or clearly progresses toward) the real resolution.

WHEN TO REJECT -- return {"recast_ok": false, "reject_reason": "<reason>", "task": "", "
steps": []}:
- Pure prose / opinion / definition / spec, or a prescriptive how-to / SOP with no real
executed actions and real results.
- Advice-only discussion (others telling the asker what to do) with no real action->
observation loop.
- Fewer than two genuine action->observation cycles, or you would have to invent the core
 facts / outcome to make one.
reject_reason is one of: no_real_observations, static_or_prose,
prescriptive_sop_no_outputs, discussion_or_advice, too_few_steps, other. Rejecting is the
 correct answer for many inputs; NEVER fabricate just to pass.

LENGTH: produce a substantial, realistic trajectory (several real action->observation
cycles, with real reasoning and real outputs), but keep the whole rendered trajectory
within about 7000 tokens. Be realistic, never pad.

The user message is the source text. Output only the JSON object.
```

I.3 Observation Audit

Gate three. The judge sees observation text alone, with no access to the source record, the trajectory around it, or the compiler's own account of what it wrote.

Prompt for observation audit

```
You are auditing training data for tool-using agents.

You will be given several OBSERVATIONS. Each one was recorded as the output of a tool
call inside an agent trajectory. For each observation, decide what it actually is:

"machine" - it is the literal output that a program, command, API, compiler, database,
browser, or service would emit. This includes logs, stack traces, error messages, result
sets and tables, JSON/dict/array dumps, file contents, source code, config, status lines,
 progress lines, version banners, and empty or no-op output. Machine output in any human
language is still "machine".

"narration" - it is a person or an assistant DESCRIBING what happened rather than the raw
 output itself. This includes: summarising a result in prose, giving advice or
recommendations, asserting that something worked, explanatory text written as code
comments, and invented confirmation messages for actions that produce no textual output
at all (for example GUI clicks, drag-and-drop, or clicking a button in a web console).

Disambiguation for code: file contents, source code, or config returned by a read/cat/
open tool is "machine". But a response made up only of comment lines that explain what
happened or what a change implies - for example several lines each starting with // or #
and containing no executable statements - is "narration". No tool emits a bare commentary
 block.

Disambiguation for logs: many real log lines are written as full English sentences, and
they are still "machine". If the text carries any emitter marker - a level prefix such as
 INFO:, WARNING:, ERROR:, SEVERE:, DEBUG:, a logger or class name, a timestamp, a thread
or process id, or key=value fields such as sourceId=... or severity=... - then it was
emitted by a program, however conversational it reads. Only call it "narration" when the
sentence is the assistant's own account of what happened, with no such marker.
```

```
Decide only on the FORM of the text: would a machine have emitted exactly these bytes? Do
 NOT judge whether the content is correct, useful, complete, or well formatted. A short
or ugly output is still "machine". A fluent, well-written summary is still "narration".

Return ONLY a JSON object of this exact shape, with one label per observation, in the
same order as given:
{"labels": ["machine", "narration", ...]}
No prose, no markdown, no explanation.
```

I.4 AnchorStretch

Four prompts. The first selects a candidate split from a trajectory, the second reviews it, and the last two do the same for the probe construction that supplies the remaining quarter of mains. The reviewers decide on their own scores and never on the score the selector assigned, which is what makes the contrast in Section B a contrast between self-report and independent review.

Prompt for AnchorStretch extractive selection

```
You are selecting REAL long-range dependencies from an existing agent trajectory.

You receive:
1) one ANCHOR_KEY already verified to occur in the trajectory; and
2) the same trajectory split into numbered, complete blocks.

Choose ONE LATER block whose reasoning, tool choice, argument, correction, or final
conclusion genuinely depends on
remembering ANCHOR_KEY. The chosen block and everything after it already exist in the
real trajectory; you are ONLY
selecting a split point. Do not rewrite, extend, repair, or invent any action,
observation, value, file, result, or answer.

Return ONLY one strict JSON object:
{"ok": true,
 "resume_block_index": <integer>,
 "dependent_quote": "<20-160 characters copied VERBATIM from the selected block>",
 "dependency_type": "<parameter_reuse|method_reuse|constraint_recall|error_interpretation
|failed_path_avoidance|state_recall|other>",
 "dependency_score": <integer 0-5>,
 "dependency_note": "<one concise sentence explaining why predicting this later block
needs the earlier anchor>"}

HARD RULES:
- resume_block_index must be strictly after the block containing ANCHOR_KEY.
- Select only THINK or PLAIN_TEXT (a user-facing/final block). Never select TOOL_CALL or
TOOL_RESPONSE:
  resuming at TOOL_CALL would split an existing think->action pair, while TOOL_RESPONSE
is supplied by the environment.
  The downstream packer inserts distractors immediately before this block, so the split
must be a safe task-resume boundary.
- dependent_quote must be copied exactly from that selected block.
- The dependency must be causal, not merely topical. If the later block could be produced
 equally well without seeing
  ANCHOR_KEY, reject it.
- dependency_score=5 means the exact later decision/value is impossible to recover
reliably without the anchor;
  4 means the anchor materially disambiguates it; <=3 must be rejected.
- Do not use the separately generated follow-up probe as evidence. Judge only the
original numbered trajectory blocks.

If no genuine existing dependency is present, return:
{"ok": false, "reject_reason": "<no_later_dependency|anchor_too_late|
trajectory_too_shallow|other>",
 "resume_block_index": -1, "dependent_quote": "", "dependency_type": "", "
dependency_score": 0,
 "dependency_note": ""}

Rejecting is preferable to inventing a dependency. Output JSON only.
```

Prompt for AnchorStretch extractive review

```
You are a STRICT counterfactual reviewer of an EXTRACTIVE long-range dependency.

The trajectory text is immutable. An earlier ANCHOR and a later RESUME block were
selected from the same real
trajectory. A packer will insert tens of thousands of unrelated tokens immediately before
 RESUME.

Judge whether predicting RESUME really requires the exact earlier ANCHOR, rather than
being recoverable from
RESUME itself or generic domain knowledge.

Return ONLY one JSON object:
{"dependency_necessity": <int 0-5>,
 "local_answerability": <int 0-5>,
 "resume_coherence": <int 0-5>,
 "anchor_specificity": <int 0-5>,
 "keep": <true|false>,
 "reason": "<=40 words"}

Scoring:
- dependency_necessity: 5 = the exact later value/choice/diagnosis cannot be recovered
without the anchor;
  4 = the anchor materially disambiguates it; 0-3 = topical, generic, or weak.
- local_answerability: 5 = RESUME/local suffix already states enough to derive the answer
 without the anchor;
  0 = impossible locally. LOWER is better.
- resume_coherence: 5 = RESUME starts at a natural THINK/plain-text continuation after a
long interruption;
  0 = broken context, dangling reference with no recoverable state, or unsafe boundary.
- anchor_specificity: 5 = concrete unique value/name/decision/root cause; 0 = vague topic.


Set keep=true ONLY if dependency_necessity>=4, local_answerability<=1,
resume_coherence>=4, and anchor_specificity>=3.

Do not trust the supplied dependency note or original score; independently test the
counterfactual:
"Could a competent agent produce this exact RESUME decision if the ANCHOR EVIDENCE were
hidden?"
Rejecting weak dependencies is preferred. Output JSON only.
```

Prompt for AnchorStretch probe construction

```
You are building LONG-HORIZON agent-training data. You are given ONE already-clean agent
trajectory ("the main task"): alternating lines of `User:` / `<think>...</think>` / `<
tool_call>{...}</tool_call>` / `<tool_response>...</tool_response>` / plain final text,
describing how a real multi-step task was actually solved.

Your job: plant ONE *verifiable long-range dependency* on top of this main task. Later a
mechanical packer will insert tens of thousands of tokens of UNRELATED distractor content
 in the middle; to finish correctly the model will then be forced to recall a specific
EARLY fact. You produce the two things that require judgement:
  1) ANCHOR  -- pick ONE concrete fact established EARLY in the main task that a later
step could genuinely depend on (a real value / name / decision / root-cause, e.g. "sys.
stderr", "max_connections=200", "the deadlock is in flush()"). Give a SHORT verbatim
handle for it.
  2) BACK-REFERENCE PROBE -- a NEW, short sub-goal that goes at the very END; doing it
correctly REQUIRES recalling that anchor from far above. Its reasoning must explicitly
name the anchor.

Return ONLY one JSON object, no markdown, no commentary:
{"ok": true, "reject_reason": "",
 "anchor_key": "<SHORT exact string copied VERBATIM from the main task -- the concrete
fact the probe depends on>",
 "anchor_quote": "<the one line/sentence from EARLY in the main task that establishes
this fact>",
 "anchor_why": "<one line: why this is a concrete, reusable anchor>",
 "probe_user": "<the new sub-goal the user asks at the END. It MUST NOT restate the
anchor value. Phrase it so the model has to remember, e.g. 'now apply the same fix to the
 other module' / 'do the same for the staging config' / 'wire the second handler the same
 way'>",
 "probe_steps": [ <step>, <step>, ... ],
 "dependency_note": "<one line: why the probe is UNANSWERABLE without recalling the
anchor>"}
```

```
Each <step> is exactly one of:
{"t":"think","text":"<agent reasoning that EXPLICITLY recalls the anchor by its exact
string, e.g. 'the same fix I found earlier -- <anchor_key> -- applies here, so I need to
...'>"}
{"t":"action","name":"<snake_case tool: run/read_file/write_file/edit_file/execute_code/
query_db/call_api/test/...>","arguments":{ <real params> }}
{"t":"observation","text":"<the LITERAL machine output this step produces, consistent
with reusing the anchor -- reads like real stdout/error/return value, never a human
summary>"}
{"t":"say","text":"<optional short user-facing line>"}
{"t":"final","text":"<final answer to the sub-goal>"}
Output STRICT JSON only -- escape newlines/quotes inside strings (important for code/logs
).

HARD RULES (break any one -> set ok=false):
- anchor_key MUST be copied VERBATIM from the main task and be SHORT (a handle, not a
paragraph). It must be a concrete FACT, established EARLY, not a vague topic.
- probe_user MUST NOT contain the anchor value (no leaking) -- the whole point is the
model must REMEMBER it.
- probe_steps MUST contain: at least one `think` that recalls anchor_key by its exact
string, at least one `action`, at least one `observation`, and a `final`.
- The probe must be GENUINELY DEPENDENT: someone who never saw the anchor could not
produce the correct probe result. If it can be answered from its own local text, it is
invalid -> ok=false.
- observation must read like real machine output (error+code / stack trace / stdout /
return value / file contents / query result). NEVER a paraphrase, opinion, thanks, or
social remark. Reconstruct the literal output faithfully; invent NO new outcome or
numbers.
- Keep the source language of the main task.
- Fabricate NOTHING about the main task. The probe is a plausible EXTENSION that reuses
the anchor; all its facts must be consistent with the main task.

WHEN TO REJECT -> {"ok": false, "reject_reason": "<one of: no_reusable_anchor,
main_too_shallow, cannot_make_dependent, other>", "anchor_key":"", "anchor_quote":"", "
anchor_why":"", "probe_user":"", "probe_steps":[], "dependency_note":""}:
- The main task has no single concrete early fact worth recalling (too short / too
generic / trivially one-shot).
- You cannot design a sub-goal that truly needs the anchor without restating it.
Rejecting is the correct answer for many inputs; NEVER fabricate a dependency just to
pass.

The user message is the main-task trajectory. Output only the JSON object.
```

Prompt for AnchorStretch probe review

```
You are a STRICT reviewer of planted long-range dependencies for agent-training data. You
 are given three things: the MAIN TASK (summary), the ANCHOR (a short fact from early in
the main task), and the BACK-REFERENCE PROBE (a tail sub-goal + its steps that is
supposed to require recalling the anchor).

Judge whether this is a GOOD, verifiable long-range dependency. Return ONLY one JSON
object, no markdown:
{"faithful": <int 0-5>, "dependency": <int 0-5>, "readable": <int 0-5>, "keep": <true|
false>, "reason": "<=30 words"}

- faithful: the probe's facts / outcome are consistent with the main task and the anchor;
 every observation looks like real machine output; nothing is fabricated. 5 = fully
faithful, 0 = invents facts/outcomes.
- dependency: doing the probe correctly REQUIRES recalling the anchor from far above; it
is NOT answerable from the probe's own text. 5 = impossible without the anchor; 0 = the
probe restates or locally contains the answer (fake dependency).
- readable: the think that recalls the anchor reads like natural, correct agent reasoning.

- keep = true ONLY if faithful>=4 AND dependency>=4 AND readable>=3.
Output only the JSON object.
```