
TRANSECT: RETAINING OBSERVABILITY FOR LONG-HORIZON LLM AGENT EVALUATIONS

A PREPRINT

Toby D. Pilditch^{1*} Konstantinos Voudouris¹
 Alexandra Abbas² Cozmin Ududec¹

¹UK AI Security Institute

²Meridian Labs

ABSTRACT

Frontier AI evaluations increasingly use open-ended, agentic, long-horizon tasks whose transcripts can span hundreds of pages of outputs and actions from complex multi-agent networks. The *observability envelope*—the range of what evaluators can reliably infer about an agent’s behaviours—is therefore narrowing. Language model assistants can help classify and interpret agent behaviour but also afford human evaluators significant analytical degrees of freedom, threatening the reproducibility and auditability of language-model-based transcript analysis. Transect is an open source package built on Inspect Scout to help evaluators understand how a long agent run unfolded, identify behaviour worth investigating, and check interpretations against the transcript. Users specify task context and behavioural vocabulary in a reusable evaluation-family configuration, with judge models and analysis settings supplied separately. Transect’s navigable reports align recorded events, token use, sub-agent activity, and model-generated behavioural labels on a common turn-based timeline. Reviewers can quickly grasp a run’s narrative, trace any label or event to its source turns, and export the underlying data tables for cross-run analysis. We demonstrate the workflow on an AI R&D evaluation that generated almost 13 million tokens, dividing the agents’ work into behavioural phases aligned with research-skill classifications, sub-agent delegations and interactions, and token use. The combined view shows a focus on operational work and manuscript production, with little evidence of a sustained hypothesis generation stage—arguably a necessary component for high-quality scientific outputs. Transect’s flexible, customisable transcript-analysis pipeline will enable evaluators to keep pace with longer, more complex, more frequent AI evaluations while supporting scientific rigour, transparency, and reproducibility.

Keywords LLM evaluation · transcript analysis · observability · AI safety · agentic evaluation · expert elicitation

1 Introduction

Three shifts are reshaping frontier AI evaluation. First, static benchmarks continue to saturate, limiting their usefulness for distinguishing the capabilities of frontier AI systems [1].² Evaluation has therefore moved towards open-ended, agentic, long-horizon tasks in which AI agents plan, use tools, delegate to sub-agents, and act over many steps towards a goal, producing millions, sometimes *billions*, of tokens along the way [2–4]. Second, increasingly capable agents are arriving more frequently, leaving less time to scrutinise each system before the next appears, and agent capabilities can improve substantially between model releases [5]. Third, an agent’s measured performance can depend substantially on its token budget and opportunities to revise its work [6, 7]. Evaluations are consequently becoming longer, more

*Corresponding author: tooby.pilditch@dsit.gov.uk.

²Note that henceforth we use the term *AI model* to refer to the static set of network weights, *AI agent* to refer to a model that has access to a set of tools in an environment and can perform actions with them, and *AI system* to refer to the general infrastructure around and including that agent, including synchronous monitors, action classifiers, and multi-agent harnesses.

frequent, and more compute-intensive, imposing a growing burden on the human evaluators monitoring progress at the frontier.

We call the range of attributes an evaluator can reliably see or infer from an evaluation its **observability envelope**. This includes what an agent can do under the evaluation conditions, how dependably it performs, where and how it fails, and what propensities or dispositions it displays. The envelope depends on the evaluation apparatus as a whole: the transcript, the details of the AI system being evaluated (e.g., harness design), the transcript analysis methods, and those interpreting the evaluation evidence. There are at least four pressures on this observability envelope, increasing the risk that evaluators draw unjustified conclusions from transcripts.

First, **within-run opacity**: a final score summarises performance on a long-horizon task but obscures discarded strategies, course corrections, recoveries, unexpected behaviors, and attempts to game the evaluation [8]. Second, a **human attention bottleneck**: reviewing hundreds or thousands of pages of agent activity requires scarce subject-matter expertise and the ability to connect and co-interpret complex events at distant points in a transcript [9]. Reviewer time cannot grow in proportion to transcript volume and model release cadence [10, 11]. Third, **insufficient statistical power**: long agent evaluations are costly in time and resources, so evaluators can afford relatively few repetitions per agent and setup, leaving measurements based on final scores alone vulnerable to task variance, stochasticity, and isolated qualitative anecdotes [12, 13]. Fourth, **setup attribution uncertainty**: observed performance is determined by both the model and the system it is embedded in [14, 15]. Without observing an agent’s entire behavioural trajectory, it can be hard to assess the construct validity of an evaluation—whether the evaluation measures the intended capability rather than an artefact of its instruments or task setup [16].

Multi-agent evaluations further intensify these pressures on observability. Running agents in parallel can increase transcript volume within the same time budget, leaving more activity for reviewers to inspect. Interactions across agents and shared resources can make the course of the evaluation harder to follow. If an evaluation uses more total compute, fewer repetitions may be affordable within a fixed compute budget. Additional choices about multi-agent delegation and coordination can also make it harder to separate the effects of the model and its evaluation setup.

The evaluation community is aware of these pressures and has been building tooling to maintain and widen the observability envelope. Inspect AI and Inspect Scout provide primitives for recording, scanning, and querying evaluation traces, including with large language models [17, 18], while platforms such as Docent support interactive exploration of behavioural patterns across agent transcripts [19]. Another prominent approach is to use language model assistants with custom judging rubrics to aid transcript analysis [20]. However, such approaches introduce substantial researcher degrees of freedom, such as the judge model, rubric, unit of analysis (tool calls, chain of thought, chains of actions), and statistical analyses performed on top of these judgments. Analysts must therefore navigate a garden of forking paths in which defensible-seeming decisions accumulate into potentially unreproducible findings [21–23]. Model-assisted analysis can also make the evaluation *more* opaque rather than less: a judge produces yet more tokens to interpret, can overstate confidence, and its errors can be difficult to verify [24, 25]. If findings about agent behaviour are to be trusted, these methods must be reproducible and the effect of arbitrary analytic choices must be measured. Otherwise, the tools intended to widen the observability envelope risk narrowing it further, replacing unread transcripts with unverifiable claims.

We present **Transect**, a transcript analysis package designed to address within-run opacity and the human attention bottleneck. Its shared, source-linked account helps evaluators understand how a run unfolded, identify behaviour worth investigating, and check their interpretations against the transcript. Reviewers can also use this account to investigate setup attribution uncertainty, although separating the effects of the model and its evaluation setup requires further evidence. Its structured outputs also support comparisons across runs, though they do not resolve insufficient statistical power when too few independent evaluations are available.

The analyst supplies a reusable configuration for the evaluation family detailing the information they need to extract, along with LLM judge rubrics and model specifications that will help them do so. Built on Inspect Scout [18], Transect analyses Inspect evaluation logs [17] or supported OpenClaw exports [26]. It extracts recorded events and uses the selected judges to classify activities, producing a report organised along a shared timeline and structured tables for further analysis. Reviewers can follow the course of the evaluation, examine activities alongside token use and sub-agent deployments, and follow labels and events directly back to the source in the transcript without assembling separate scanner outputs themselves. This connects the overall account of an evaluation to the details needed to investigate an unexpected behaviour, a possible safety incident, or a constraint imposed by the evaluation setup.

Reproducible analysis requires retaining the source transcripts, run selection, family configuration, judging settings, custom analysis code and stored results, and recording software and model versions where available (Section A.7). Independent evaluators can re-examine stored results without new judge calls, although rerunning the judges may produce different labels. The shared judging machinery records judge identities and decisions, with diagnostics

of within-model stability or between-model agreement where the judging regime supports them. Of course, this information alone does not establish whether the labels are correct. These records and checks support Transect’s aim of scaling oversight responsibly: increasing analytical capacity while preserving the means to assess its findings.

The remainder of the paper describes Transect’s pipeline and design principles (Section 2), then illustrates the workflow on a long-horizon AI R&D shadow evaluation with extensive sub-agent activity (Section 3). The case examines research activities and token expenditure, uses custom scanners to trace shared-file accesses across agents, and combines scanner outputs into additional report views. A full technical reference appears in Section A, and the implementation is openly available at <https://github.com/AI-Safety-Institute/transect>.

2 Transect

Transect turns an evaluation transcript into a navigable account of the agent’s activities with direct links to their source. We describe the pipeline (Section 2.1) and the principles guiding its configuration and interpretation (Section 2.2), then work through one AI R&D run to show how a reviewer can use the report (Section 3).

2.1 The pipeline

Transect builds on Inspect Scout’s transcript and scanner primitives [18] (Section A). The pipeline starts with a completed run transcript and a **Spec**: a reusable configuration for an evaluation *family* (e.g., a cyber-security range or an AI R&D evaluation). A Spec is used for every evaluation family, whether or not the analysis calls language models. It can contain task context, activity definitions, sub-agent role categories where relevant, and user-defined content for custom analyses. Its contents vary with the task and the user’s questions: sub-agent categories, for example, are needed only when classifying sub-agent roles. The package includes skills for coding assistants to guide Spec authoring and refinement. The user starts from the evaluation brief and setup materials, identifies missing information, and seeks clarification from evaluation designers or subject-matter experts. For classification, the user supplies the categories and definitions, and language model scanners use them to label the transcript. The user can also add custom structural scanners to extract recorded information and custom judged scanners to classify behaviour (Section A.6).

The broader pipeline configuration also includes separately supplied analysis settings, such as judge models and whether and how they may interact (see Section A.1). Transect supports one pass by a single model (solo), repeated passes by that model (k -rolls), or a cohort of models making separate judgments (Section A.5). These choices specify how to analyse the run, while the Spec describes evaluation content and context. Transect retains the supplied judging settings in its stored scanner configuration, alongside records of the judging regime and results, for later inspection (Section A.7). Transect returns a `TransectResults` object containing **dataframes**, the structured tables underlying a navigable report (Section A.3).

The report helps reviewers locate episodes that warrant inspection, compare token use across activities, and investigate how recorded events relate to changes in behaviour. For example, reviewers can examine whether a shift in activity occurs near a context compaction and follow both back to the transcript. These features address the **human attention bottleneck** and **within-run opacity**, and help investigate how the evaluation setup shaped a run (Section 1).

Figure 1 shows the five stages. **Ingest** places the run on a shared **turn** axis, with one turn per agent output, so later signals use the same index across source formats. The axis records output order; the elapsed time between outputs can vary. **Select** chooses the sample and one or more epochs to analyse.³ **Scan** runs the scanners. **Store** assembles their results into Transect’s dataframes. **Render** produces the report, linking scanner outputs to their source turns in the transcript. Users can export the returned dataframes and combine them across epochs for their own analyses.

A **scanner** extracts information or classifies behaviour from a transcript. In Figure 1, *structural* scanners extract recorded information, such as context compactations, operator messages, and per-turn token use, without calling a language model. *Judged* scanners call the language models selected for the analysis to classify, rate, or annotate the evaluated agent’s behaviour. The AI R&D analysis below combines recorded operator messages and context compactations with research-activity labels from a custom judged scanner (Figure 2).

The built-in *judged* scanners run when judge models and the required Spec vocabularies are supplied. The `decision_phases` scanner divides the run into contiguous phases and labels them using a *closed, pre-registered vocabulary* based on the Spec, reusable across samples and epochs of the same family. The `subagent_classification` scanner labels sub-agents from their delegation instructions: what the orchestrator asked them to do, which may differ from what they actually did.

³A *sample* is a task instance with a specified prompt and evaluation setup; an *epoch* is one attempt at that sample.

Users can add *custom structural* or *judged* scanners to address further evaluation questions. For bespoke classification questions, the public `cohort_llm_scanner` helper gives a custom scanner the same judging machinery as the built-in scanners: voting across repeated judgments or multiple judges and a per-judgment provenance record identifying the judges, their individual answers, and any verifier changes. In solo and k-roll regimes, a verifier model can review selected classifications in an optional second round and revise their labels. Selection uses low-confidence or low-agreement triggers and random spot-checks. Nonetheless, these outputs remain fallible model judgments. Depending on the judging regime, they carry a model’s stated confidence, within-model stability, or between-model agreement (Section A.5). While stability and agreement help assess the reliability of a judgment or classification, they do not establish validity, as that would require comparison to a ground truth (usually, a human expert’s validated judgement). The reviewer ultimately remains responsible for judging what the evidence supports.

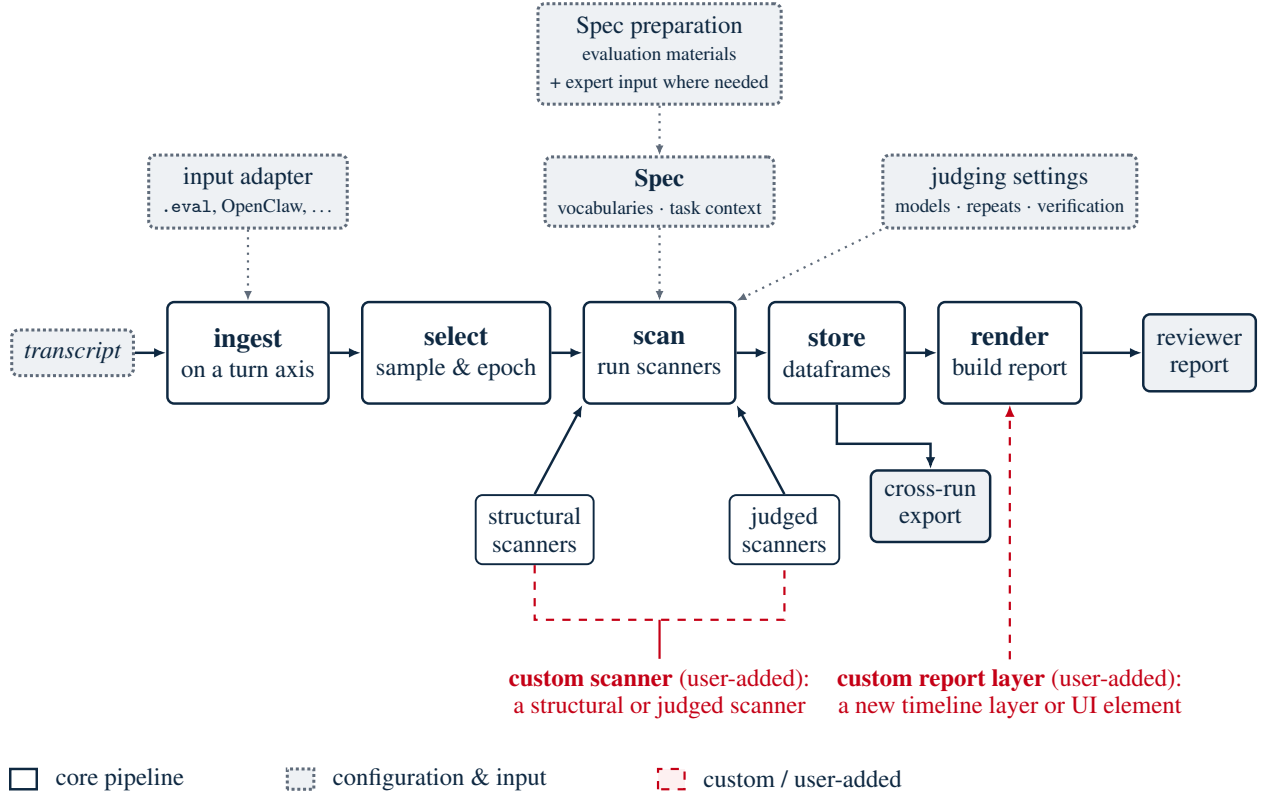


Figure 1: The Transect pipeline: **ingest**, **select**, **scan**, **store**, and **render**. The dataframes are used to generate the navigable report but can also be used for downstream cross-run analyses. Grey dotted boxes show inputs and configuration; the *Spec* draws on evaluation materials and, where needed, expert input. *Structural* scanners extract recorded information without model calls; *judged* scanners call the models selected for the analysis. Red dashed marks indicate two extension points. *Custom scanners* add *structural* or *judged* signals to dataframes. *Custom report layers* use those dataframes to add timeline layers or interface elements to the report without changing the upstream analysis.

2.2 Design principles

Three design principles guide how the analysis is configured, interpreted, and extended. The first is to configure analysis for the evaluation **family**. Each family’s **Spec** can define the behavioural vocabulary and any relevant sub-agent categories used to interpret its runs, drawing on evaluation materials and, where needed, subject matter expertise.

The second is structured expert elicitation [27, 28]. Users can ask a coding assistant, guided by the package’s skills, to flag missing task context or unclear category definitions and help prepare questions for subject-matter experts (SMEs). Experts can then supply information missing from the evaluation materials, such as outcomes of interest, and review proposed classification rubrics. Transect records the category labels and definitions used by its built-in judged scanners. These records let users inspect the applied rubric and compare revisions across analyses.

The third is automation. The package’s skills and worked examples guide coding assistants in running analyses and adding scanners or report views. They are designed to reduce the work needed to adapt the pipeline as evaluation tasks

change. For a new family, users can supply a different Spec and add analyses while reusing the existing pipeline for supported transcript formats. The diagnostics skill helps users inspect disagreements, check the recorded judging setup and test sensitivity to the Spec’s category names while keeping their definitions fixed.

A custom report layer draws on the same dataframes to add a new view (Section A). A design requirement is that report views preserve the available diagnostics and provenance of judged outputs. The report displays available confidence and agreement diagnostics and flags potential reliability problems. Provenance information helps reviewers understand the source of scanner judgements, and reliability information helps them assess what conclusions these judgements can support.

2.3 Related work

Transect builds on Inspect AI for evaluation-log access and judge-model interfaces, and on Inspect Scout for transcript representation, scanner execution and stored results [17, 18, 29]. Inspect Viz supplies the report’s interactive charts.⁴ Transect organises these components around an evaluation family and enables co-interpretation of multiple scanner outputs on a shared timeline.

Docent is another prominent transcript analysis tool which uses language model assistants to help users interactively navigate through transcripts [30]. Transect, in contrast, supplies a reusable analysis and reporting pipeline: users can apply a family Spec and their chosen judging settings across samples and epochs. The package’s coding-assistant skills help users add analyses or views for new questions while reusing shared tables and report interfaces. Platforms such as LangSmith, Langfuse, Arize Phoenix, Weights & Biases Weave, and AgentOps support trace inspection, debugging and application monitoring; LangSmith and Langfuse also support offline experiments and online evaluation.⁵

Transect brings family-defined activity labels, recorded events and token use together on an interactive turn-level timeline, so reviewers can examine how these signals relate within a run and follow them to the source transcript. Transect’s contribution is to combine adaptable analysis, interactive evidence review and inspectable model judgements in one reusable workflow for long-horizon evaluations. Methods for statistical analysis and evaluation design are complementary [12, 31], since Transect structures within-run evidence for such analysis but does not make sparse observations statistically sufficient.

3 Case Study: Reading an AI R&D Evaluation with Transect

AI research and development (R&D) is a prototypical example of an open-ended, long-horizon task for frontier agents. AI R&D evaluations measure whether current agents can perform machine learning research that might accelerate AI development, including through so-called *recursive self-improvement* [32–35]. Such evaluations exemplify the difficulties of maintaining observability on agent behaviour. An agent may operate autonomously for days or weeks, generate billions of tokens, execute thousands of tool calls, and delegate work to hundreds of sub-agents [36]. It is also unclear what successful behaviour looks like, since “doing good science” is a fuzzy construct [37–40], and the final artefact is a written paper or report, whose quality may conceal many issues during its construction, including data fabrication, citation hallucination, and deep conceptual misunderstandings [41–43].

We illustrate Transect on a single sample of the CRUX initiative’s *shadow evaluations* [44]. In these evaluations, an agent receives a research brief based on a complete but unpublished human-written machine-learning research paper and is asked to produce a finished manuscript. A human operator is available over a Telegram chat interface to resolve operational problems but does not direct the research process itself.

The brief for this sample poses a methods problem in tabular prior-fitted networks (PFNs), transformers that classify tabular data through in-context learning over a support set rather than task-specific training. The research agent was asked whether a deployment-time detector could identify *harmful* distribution shifts (changes that degrade the model’s accuracy) without erroneously flagging benign shifts and without access to labels at inference time. The brief given to the agent requested a method to solve this problem which outperforms a set of pre-specified classical shift detector baselines and survives a pre-registered falsification test. The agent had to write and submit a manuscript that it had reviewed and deemed worthy of acceptance at a machine learning conference, along with a package for reproducing the analyses and results.

⁴https://meridianlabs-ai.github.io/inspect_viz/.

⁵<https://docs.langchain.com/langsmith/evaluation>; <https://langfuse.com/docs/evaluation/overview>; <https://arize.com/docs/phoenix>; <https://docs.wandb.ai/weave>; <https://docs.agentops.ai/>.

The resulting OpenClaw transcript [26] sees a Claude-Opus-4.7 agent [45] produce some 554 orchestrator tool calls and 79 sub-agent deployments, receive 57 operator messages, and undergo 8 context compactions. 13 million tokens were generated during the evaluation, 89% of them in delegated sub-agent work.

Our purpose is not to provide a complete assessment of the agent’s AI R&D capability, for which we refer the reader to Kirgis et al. [44]. Instead, we ask four questions a reviewer of this run would want answered: how the research process unfolded (Section 3.1), where test-time compute was spent (Section 3.2), how delegated agents coordinated (Section 3.3), and what further questions can be asked by co-interpreting scanner outputs (Section 3.4). Each subsection reports an observation, discusses plausible interpretations, and notes how Transect produced it. Section 3.5 then discusses what the case establishes and what it does not.

3.1 How did the research process unfold?

Figure 2 places six layers on a shared **orchestrator-turn** axis, where one turn corresponds to one orchestrator output, so that the layers can be read against one another. The package’s shared model-turn axis assigns an index to each recorded orchestrator model output (Section A.2). The lower layers come from *structural* scanners, which read facts already recorded in the transcript without calling another model (in ascending order): how the orchestrator’s context changed across compactions, when the operator sent a Telegram message, when sub-agents were launched and returned, and, from a custom scanner that identifies arXiv identifiers, DOIs and heuristically matched titles, where references first appear. Sub-agent markers are labelled using the built-in `subagent_classification` scanner, which describes what the orchestrator asked each sub-agent to do, not necessarily the work it ultimately performed. The two upper bands come from custom *judged* scanners that assign one principal label, or a residual “Other”, to each of the epoch’s 580 orchestrator outputs, including tool-only outputs. The research-activity band uses ten named categories intended to capture the typical structure of AI R&D tasks: *Hypothesising*, *Literature grounding*, *Experiment design*, *Analysis design*, *Falsification*, *Research engineering*, *Debugging*, *Evidence interpretation*, *Reflecting* and *Writing up*. The paper-specific band uses the closed vocabulary defined for *decision_phases* in the family’s **Spec**, which follows the task’s concrete research questions and deliverables, from *PFN problem framing* and *PFN solution ideation* through *Benchmark construction*, *PFN model evaluation* and *Falsification tests* to *Manuscript write-up*, *Self-review* and *Code reproducibility*, and separately tracks *Non-research-relevant engineering*. The category definitions are given in Section A.4.

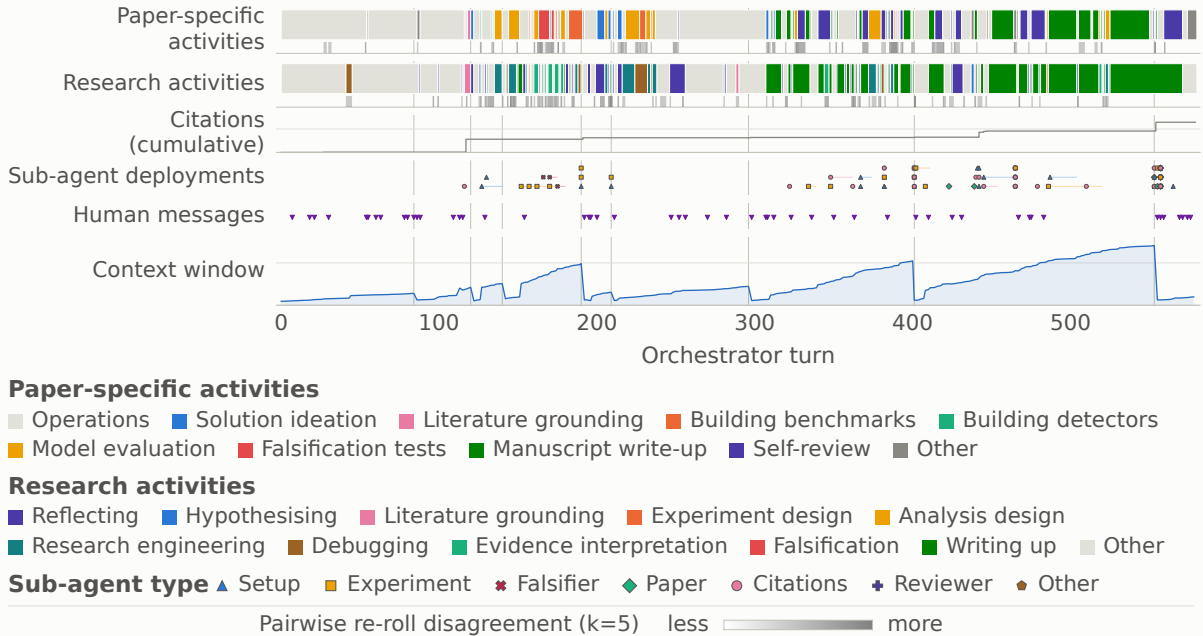


Figure 2: The TabPFN epoch on an orchestrator-turn axis: paper-specific activities, research activities, cumulative references, sub-agent deployments, operator messages, and context-window size. Grey strips show pairwise label disagreement across five judge rolls for every turn; darker means greater disagreement.

The value of the timeline comes from reading these independently produced layers together. The epoch begins with a long operations-dominated stretch containing little judged research activity other than debugging. This also coincides

with frequent operator messages, mostly helping the model to establish permissions for its various tools. Inspection of these turns shows the agent establishing the environment’s permissions, resources, and available tools. Much of the initial literature engagement follows this operational period. The middle of the epoch alternates between benchmark construction, model evaluation, and falsification, with several delegated sub-agents supporting those activities. The final portion concentrates on manuscript drafting and self-review, with writing, citation, and reviewer sub-agents used to parallelise that work. Some context compactions also occur near sub-agent deployments, exposing a possible relationship that can be further investigated rather than merely noticed while reading the raw transcript.

The combined view also makes absences salient. In the current analysis, only two of the 580 orchestrator outputs are labelled *Hypothesising*, compared with 56 labelled *Reflecting* and 27 labelled *Evidence interpretation*. Surprisingly, *Analysis design* does not appear at all. The two hypothesising labels occur when the orchestrator plans to derive novel theoretical results for a detector it has already proposed, and it delegates this work to a sub-agent both times. This pattern has at least two potential explanations: the agent may have performed little hypothesis formation, or the scanner may under-detect diffuse or secondary activities when assigning a single principal label. A missing label is therefore weak evidence that the behaviour was absent. Even with that limitation, the timeline turns the impression that “the agent rarely seems to ideate” into a reproducible observation linked to specific turns, an explicit rubric, a reliability indicator, and competing explanations for review.

Both activity bands were produced in a *k-roll* regime: Claude Opus 4.7 classified each output $k = 5$ times, reading the target’s full text and tool arguments with up to three preceding and following outputs, and the bands display the modal label with adjacent identical labels merged, forming 129 contiguous phases for the paper-specific activities. The grey strip beneath each band shows the fraction of the ten judge-roll pairs assigning different labels: 474 of 580 outputs (81.7%) receive unanimous research-activity labels and 476 (82.1%) unanimous paper-specific labels, with *Reflecting* and *Evidence interpretation* comparatively unstable. A *cohort* regime could instead call several judge models to expose between-model agreement. A *solo* run would call one model once and would therefore be cheaper, but would provide no empirical stability estimate. The stored scan results retain the prompts, inputs and model responses behind every label so that they can be inspected transparently. Variation in judgements helps identify labels that merit caution or rubric refinement, but low variation does not establish that the labels are *valid*, which would require independently collected human labels or another suitable external referent to calculate false positive and negative rates (Section A.5).

3.2 Where was test-time compute spent?

The same turn-level decomposition supports a second question: how was the test-time compute allocated? Token telemetry, activity labels, and sub-agent events share the same segmentation, so token expenditure can be attributed to the activity and the sub-agent responsible for it. Figure 3 shows the three attributions side by side.

In the current analysis, self-review and manuscript write-up account for roughly three quarters of all new-work tokens: 7.8 million and 2.9 million, respectively, out of 14.5 million. Almost all of this expenditure is delegated, reflecting the orchestrator’s use of citation, writing, and reviewer sub-agents during the final stages. Non-research engineering is the third largest category at approximately 2.2 million tokens, much of it associated with resolving permissions and determining the scope of the evaluation environment.

By contrast, solution ideation only accounts for approximately 22,000 tokens. Some relevant work may have been labeled under reflection, research engineering, or evidence interpretation, which account for substantially more expenditure. Nevertheless, this allocation is consistent with the original evaluation’s observation that agents “produced papers with extremely thorough negative findings rather than papers with new ideas” [44, p. 16]. Citation sub-agents consume approximately 5 million tokens, but much of this work occurs during manuscript drafting and self-review rather than early hypothesis development. One plausible reading is therefore that literature search primarily supports polishing and substantiating the manuscript rather than generating new scientific directions.

These findings also bear on harness and setup uncertainty. Non-research operations occupy a large fraction of the epoch, and most of the operator messages occur during those phases, often providing permissions, resources, or credentials needed to overcome an infrastructural impasse. The observed allocation could reflect an agent preference for tractable operational and writing work, a scaffold that inadequately supports scientific work, or scientific reasoning that is not exposed in the recorded transcript. Transect cannot distinguish these explanations by itself. It makes them precise enough, however, to motivate interventions: evaluators could change the initial permissions, provide a different tool set, alter the research brief, or add progress gates, and then compare the resulting phase and token distributions across epochs.

Token allocation also provides a way to study capability as a function of test-time compute. Two systems may produce similarly rated final artefacts while using compute differently: one may brute-force many weak experiments, while another may perform fewer, better-motivated tests. Likewise, apparent improvement may be concentrated in research

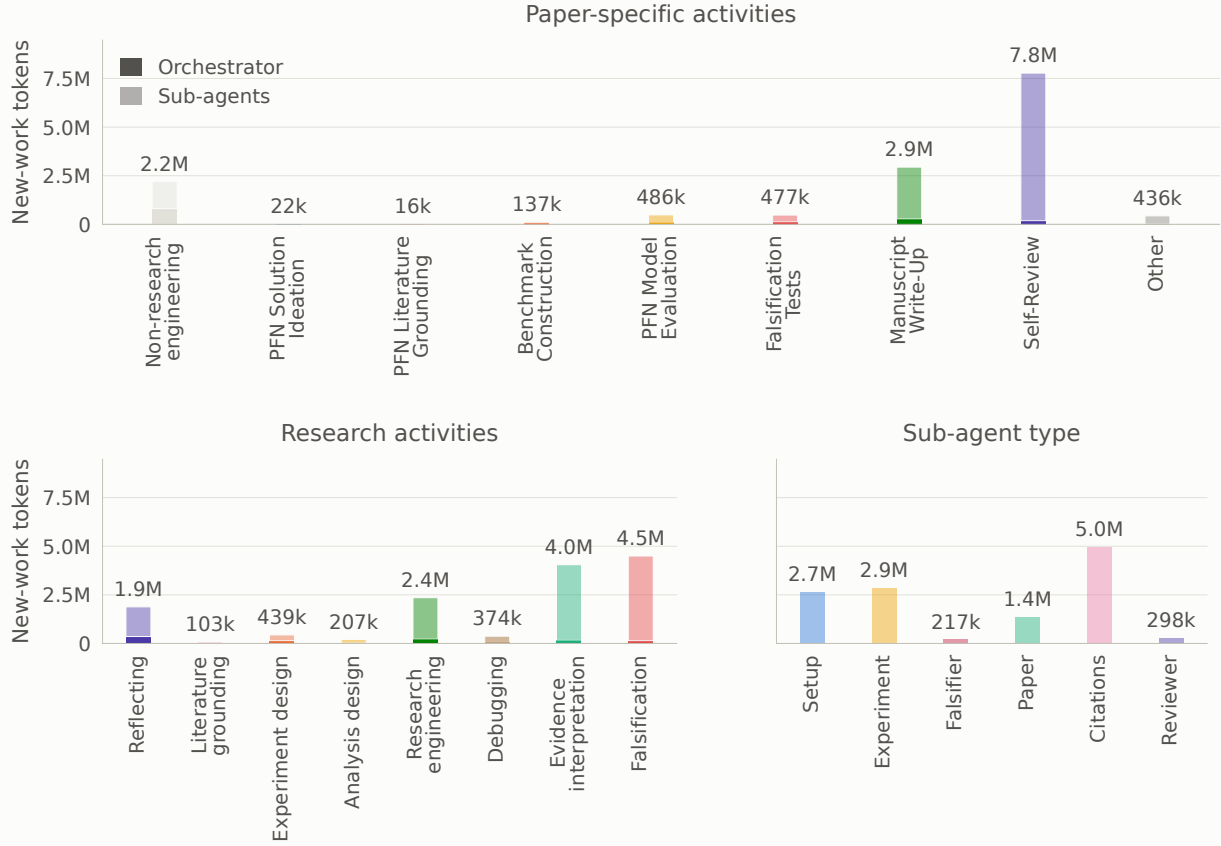


Figure 3: New-work tokens attributed to paper-specific activities, research activities, and sub-agent categories. In the activity panels, solid segments show the orchestrator’s own expenditure and lighter segments the expenditure of sub-agents launched within the unit. The research activities panel excludes work without an assigned research skill, and the sub-agent panel excludes the orchestrator’s own activity. Token counts should not be summed across panels.

engineering or manuscript production rather than in the scientific activities relevant to capability forecasts, such as solution ideation, experiment design, or falsification. One transcript cannot establish these differences, but the exported dataframes make the same decompositions available across agents, budgets, scaffolds, and repeated epochs.

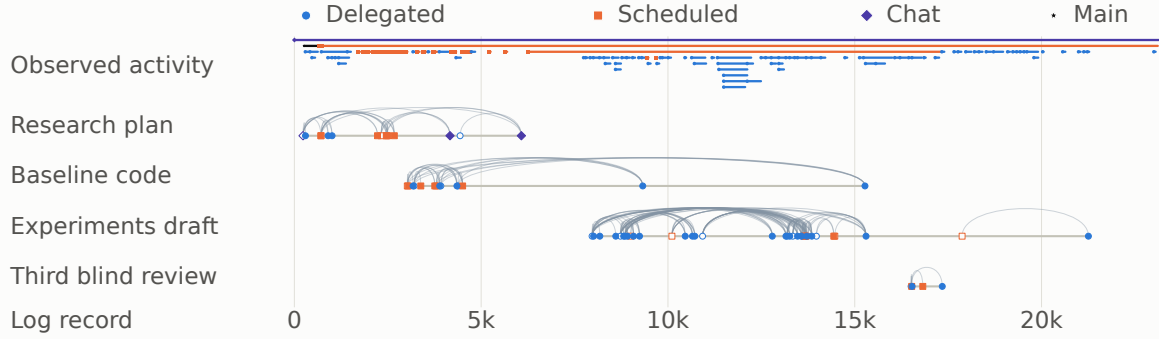
3.3 How did delegated agents coordinate?

A multi-agent evaluation creates an interaction structure that is difficult to reconstruct from raw logs: which shared files agents wrote and read, in what order, and what kinds of exchange these were. To follow work across agent boundaries we extend the report with a custom *structural* scanner for shared-file access and an optional *judged* scanner for its likely purpose, and combine their outputs in Figure 4.

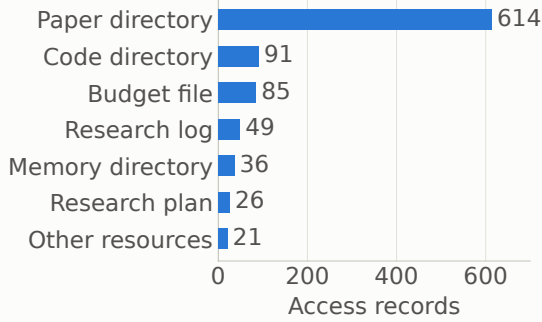
The structural scanner joins recorded file access events to earlier writes by other separately logged agent conversations or tasks (*sessions*), preserving session identifiers, resource paths, source positions, and evidence types in the **dataframes**. It produces 922 access records across 131 participating sessions, of which 346 records connect one delegated sub-agent to another. The sessions comprise delegated sub-agents, tasks triggered by the harness’s scheduler, and the harness’s main and Telegram chat contexts. The dataframes retain the writer-reader direction.

Figure 4, Panel A follows the research plan, baseline code, experiments-section draft, and third blind-review document through the recorded workflow. These four tracks represent 159 access records involving 39 sessions; the activity strip retains all 131 participants. Each arc connects a record’s first qualifying read to the named writer’s latest preceding write. Agent bars span first to last recorded activity, while file tracks span first recorded write to last recorded read or

A) Agents and shared files



B) Shared source location



C) Inferred Purpose of Access

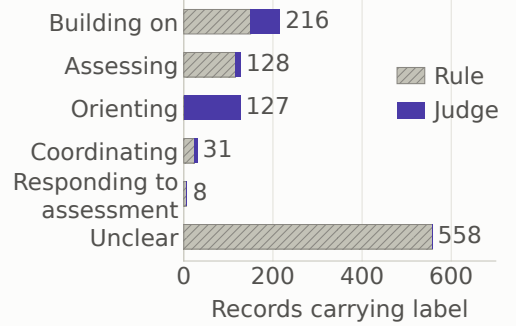


Figure 4: Multi-agent analysis of one TabPFN epoch. A: activity across 131 agent sessions and four shared-file tracks in source-log order. Arcs connect earlier writes (open symbols) to later cross-agent reads (filled symbols); colour and shape denote context type. B: all 922 access records by resource path. C: inferred purposes, with compound labels counted in each category. Access does not establish uptake.

write. The axis follows source-file order: these spans establish neither elapsed time nor continuous agent activity or file persistence.

Access is concentrated in the manuscript workspace: 614 of the 922 records (66.6%) concern paths under `paper/`, including drafts, reviews, and build files. The experiments-section draft alone accounts for 102 records, visible as repeated connections along its track. Together with the token attribution in Section 3.2, this identifies the manuscript workspace as a priority for examining how delegated contributions were assembled. A reviewer can follow a shared document to its recorded writers and readers, then inspect their instructions and subsequent edits.

Purpose classification helps reviewers follow concrete exchanges. For example, the transcript records a sub-agent opening the third blind-review document after receiving instructions to address five major and four minor presentation findings and rebuild the manuscript. All five judge rolls classify this access as *responding to assessment*. This connects the review track in panel A to a revision brief and a recorded read, giving the reviewer a concrete trail to investigate. Inspecting subsequent edits would establish which findings were addressed and how.

This remains a prototype, but its provenance, coverage, and repeatability diagnostics make the limits explicit. Across the transcript, 150 records receive substantive rule-assigned labels and 214 receive judge-assigned labels; 558 remain unclear. Among the judge-labelled records, 166 (77.6%) receive unanimous labels across five Opus 4.7 rolls. This measures within-model repeatability; purpose validity and uptake require further evidence. The coverage gaps suggest concrete improvements: parsing omitted shell commands and direct messages would capture more exchanges, while harness records of read results, file versions, and sub-agent return payloads would help reviewers trace contributions and check the inferred purposes.

3.4 What else can we learn about AI R&D?

Once phase labels, recorded events and reference records share one representation, questions that would otherwise require bespoke transcript processing become cheap to formulate and test. We ask three here: whether the workflow has

recurring structure, whether behavioural transitions cluster near operator messages or context compactions, and how the agent’s literature coverage compares with that of the human authors. Figure 5 presents descriptive analyses of these three questions.

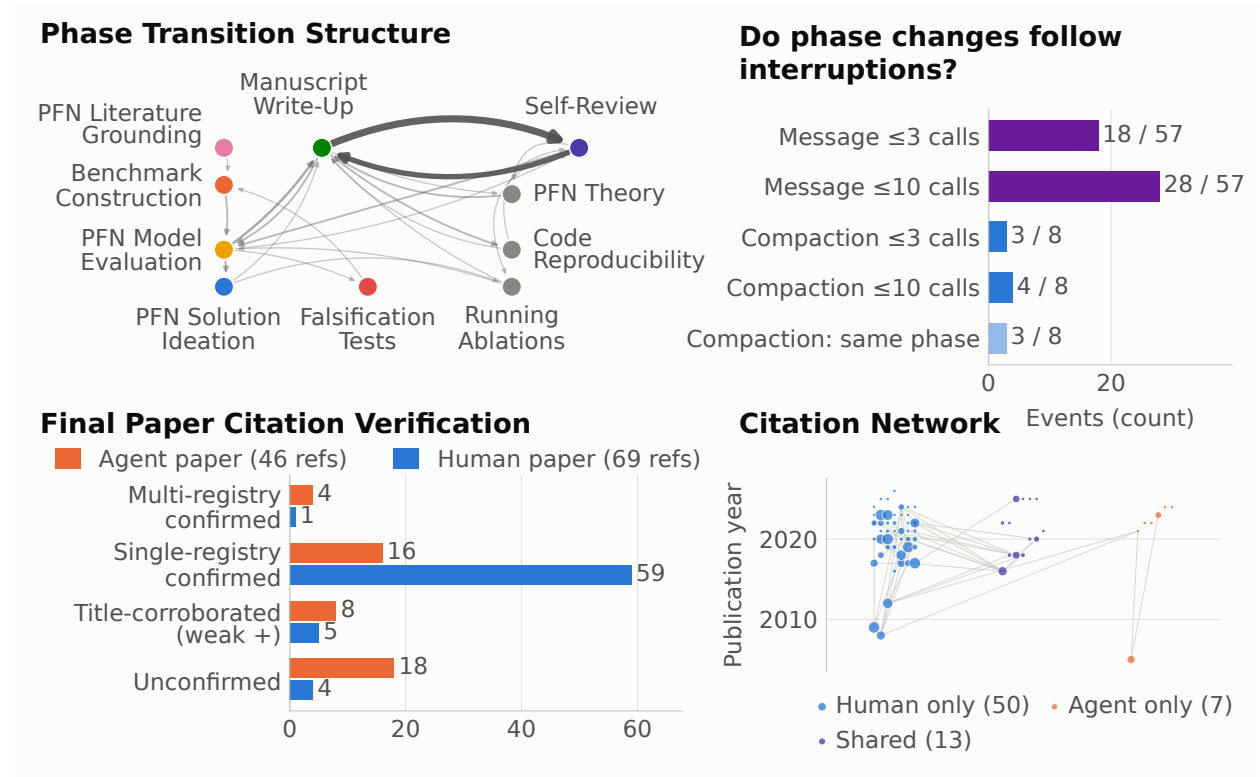


Figure 5: Descriptive analyses of the TabPFN epoch. Top left: observed transitions between paper-specific phases after removing non-research-relevant engineering phases and merging consecutive repeats, with edge width proportional to transition count. Top right: the number of operator messages and context compactions followed by the start of a new phase within 3 or 10 orchestrator tool calls, and compactions after which the current phase continues; these are co-occurrence counts, not effects. Bottom left: registry verification of references in the agent- and human-written manuscripts against OpenAlex [46], Crossref [47], and arXiv [48]. Bottom right: the combined citation network, with node size indicating global citation count and edges indicating citations among works in the combined bibliography.

The phase-transition graph shows the transitions between paper-specific research activities, excluding non-research engineering. Two regimes are visible: a build-and-test loop through benchmark construction, model evaluation, ablations, and falsification; and a write-and-review oscillation between manuscript write-up and self-review, with some input from theory building, code reproducibility, and running ablations.

This phase-transition graph (top left panel, Figure 5) illustrates how an ordered sequence of task-specific phase labels makes transitions between activities directly computable.

This graph structure can also guide investigation of the evaluation setup. The repeated cycle between manuscript drafting and review motivates testing whether the setup encourages iteration over the written artefact. The build-and-test loop is less dominant and does not show an equivalently strong earlier cycle of generating, selecting, and testing alternative ideas. Evaluation and harness designers could use this graph as a target for intervention, for example by introducing an explicit ideation gate or requiring comparison among several candidate methods before manuscript drafting begins. Comparing repeated epochs under otherwise comparable original and modified setups would help assess the intervention’s effects on behaviour and the final score.

The event-proximity panel (top-right panel, Figure 5) shows whether new phases begin near operator interventions or context compactions. In the current analysis, a new phase begins within three indexed steps of 18 of the 57 operator messages and within ten steps of 28 operator messages. New phases begin within 3 indexed steps of 3 compactions and within 10 steps of one further compaction. These are raw co-occurrence counts, not causal effects: operator messages may respond to an already emerging transition, and compactions may occur at points where the agent would have

changed direction anyway. Nevertheless, both event streams are partly determined by the evaluation setup, so their proximity to behavioural transitions identifies a concrete source of setup uncertainty for controlled follow-up.

The final two panels use a feature of the shadow-evaluation design: the agent-written manuscript can be compared with the unpublished human-written paper on which the brief was based. We first verify cited references against OpenAlex [46], Crossref [47] and arXiv [48]. 20 out of the agent’s 46 cited references are verified in at least one of these registries, and a further 8 have title matches without linked DOIs, leaving 18 unconfirmed. In contrast, 60 out of the human paper’s 69 references are verified in at least one registry, with a further 5 having a title match, leaving only 4 unconfirmed references. Registry non-matches do not prove fabrication, but the difference identifies a substantially greater citation-verification burden in the agent-written manuscript.

The citation network provides a complementary view. The manuscripts share 13 references, but the human paper draws on a larger and more internally connected literature, as indicated by the density of citation links among its referenced works. This is consistent with the possibility that the agent’s search did not recover the same scientific background as the human researchers. It also suggests a follow-up intervention: evaluators could supply the human bibliography as part of the scaffold and test whether access to the same literature changes the agent’s hypotheses, experimental choices, or final claims.

The package supports displaying analyses such as these, alongside its timeline and token allocations, as custom report elements (Figure 1, Section A.6). Linking their summaries to phases and source turns requires matching transcript identities and compatible turn or tool-call indices.

3.5 What this case establishes

This analysis establishes some interesting features of the agent’s behaviour: It starts with a long operational opening without much hypothesis generation. Most of the tokens are spent by sub-agents doing manuscript writing and review, phases through which the agent iterates several times.

However, this analysis is descriptive rather than confirmatory. It does not firmly establish the agent’s AI R&D capability, the validity of the judged labels, or causal effects of operator messages, compactions, or the scaffold. Those claims would require multiple epochs on multiple samples from the same evaluation family, independently collected expert labels for the judged scanners, and an explicit statistical model for trends across runs (Section A.3). Transect offers a coherent framework for doing this kind of rigorous, confirmatory analysis on open-ended, long-horizon evaluations, while maintaining reproducibility and transparency.

4 Conclusion

Open-ended, long-horizon agent evaluations are narrowing the observability envelope—what an evaluation reliably tells us about an agent—because they are getting longer, more complex, and more parallelised. A natural solution to this is to use language models to assist with transcript analysis, but this leads to significant researcher degrees of freedom. We present Transect as a tool to address these twin problems, by making within-run behaviour scrutable while supporting reliability checks and reproducible analysis. Transect places a run on a shared turn axis, selects the sample and epoch to read, reads the transcript with structural and judged scanners, and writes every signal into one set of dataframes that can be digested as a human-readable report. This is the heart of the value proposition of Transect: it allows evaluators to both readily access the ‘wider picture’ and quickly identify and analyse key moments in the agent’s trajectory - whether these are indicative of agent capability, safety incidents, harness constraints, or other research questions the decomposition exposes. We argue that this human-accessible, cohesive picture is critical for maintaining effective oversight for increasingly long and complex evaluation transcripts.

While language models are becoming increasingly important for transcript analysis, they must be used with caution. LLM-based scanners can overstate confidence, hide uncertainty, and ultimately be over-interpreted by human reviewers [24, 49]. Language models are increasingly capable as judges for many classification tasks, though reliability does not establish validity [25]. Transect records available reliability diagnostics and flags potential problems for review. Assessing validity requires separate evidence, such as comparison with independently collected expert annotations.

Expert elicitation, long established in risk analysis [27], can calibrate Transect through written materials or targeted questions that shape which signals a family of evaluations should surface. Reviewers can examine classifications alongside the agent’s recorded reasoning, tool use, and harness events. These signals need not be independent, but reading them together can expose gaps or inconsistencies that warrant inspection. Provenance records show how a classification was produced and which source turns it was derived from. This view can also help evaluators investigate how the harness shapes agent behaviour and where its records are incomplete. For example, if the transcript does not

record a tool failure, adding that event to future logs would let reviewers examine subsequent behaviour alongside the failure. The package’s coding assistant skills and worked examples are designed to reduce the work needed to configure new evaluation families and extend their analyses. They guide coding assistants in adding scanners or report views, inspecting judging records and diagnostics, and testing sensitivity to category names. These provenance and reliability disciplines support reproducibility and should accompany each extension of the analysis.

Transect turns long evaluation transcripts into source-linked reports and reusable analysis tables. The case study illustrates how these outputs bring behavioural classifications, token use across the orchestrator and sub-agents, multi-agent interactions, and harness events into one review workflow that is navigable, auditable, and reproducible.

Acknowledgements

We are grateful to the authors of the CRUX shadow evaluations [44] for providing the data for the case study presented in this paper. We would also like to thank Peter Kirgis, Sayash Kapoor, Andrew Schwartz, Stephan Rabanser, Viet Nguyen, Arvind Narayanan, Michael Schmatz, Giorgi Giglemiani, Jessica Wang, Pawan Mudigonda, Magda Dubois, Auss Abbood, Orazio Angelini, and Ole Jorgensen for thoughtful discussion throughout the development of Transect. Finally, we extend our thanks to Meridian Labs [50] for their close collaboration in the development of the package.

References

- [1] Lexin Zhou, Lorenzo Pacchiardi, Fernando Martínez-Plumed, Katherine M Collins, Yael Moros-Daval, Seraphina Zhang, Qinlin Zhao, Yitian Huang, Luning Sun, Jonathan E Prunty, et al. General scales unlock AI evaluation with explanatory and predictive power. *Nature*, 652(8108):58–67, 2026.
- [2] Sayash Kapoor, Peter Kirgis, Andrew Schwartz, Stephan Rabanser, J. J. Allaire, Rishi Bommasani, Harry Coppock, Magda Dubois, Gillian K. Hadfield, Andrew B. Hall, Sara Hooker, Seth Lazar, et al. Open-world evaluations for measuring frontier AI capabilities. *arXiv preprint arXiv:2605.20520*, 2026.
- [3] Asaf Yehudai, Lilach Eden, Alan Li, Guy Uziel, Yilun Zhao, Roy Bar-Haim, et al. Survey on evaluation of LLM-based agents. *arXiv preprint arXiv:2503.16416*, 2025.
- [4] Mahmoud Mohammadi, Yipeng Li, Jane Lo, and Wendy Yip. Evaluation and benchmarking of LLM agents: A survey. *arXiv preprint arXiv:2507.21504*, 2025.
- [5] Linus Folkerts, Will Payne, Simon Inman, Philippos Giavridis, Joe Skinner, Sam Devere, James Aung, et al. Measuring AI agents’ progress on multi-step cyber attack scenarios. *arXiv preprint arXiv:2603.11214*, 2026.
- [6] Jessica McFadyen, Ole Jorgensen, Harry Coppock, Kevin Wei, and Cozmin Ududec. How inference compute shapes frontier LLM evaluation, 2026. URL <https://arxiv.org/abs/2606.17930>.
- [7] UK AI Security Institute. More compute, more capability: Why AI agent evaluations need to account for test-time compute. AISI blog (Science of Evaluations team), 2026. URL <https://www.aisi.gov.uk/blog/more-compute-more-capability-why-ai-agent-evals-need-to-account-for-test-time-compute>.
- [8] UK AI Security Institute. Cheating behaviour in frontier model evaluations. AISI blog, 2026. URL <https://www.aisi.gov.uk/blog/cheating-behaviour-in-frontier-model-evaluations>.
- [9] UK AI Security Institute. Security Incident INC-2026-07-28-01. Technical Report INC-2026-07-28-01, UK AI Security Institute, July 2026. URL https://cdn.prod.website-files.com/663bd486c5e4c81588db7a1d/6a724858f7db25c81487016d_Security%20Incident%20INC-2026-07-28-01.pdf.
- [10] Patrik Reizinger and Wieland Brendel. Skills, benchmarks, and verification are what AI-assisted research needs. In *International Conference on Learning Representations (ICLR)*, 2026.
- [11] Jiachen Liu, Jiaxin Pei, Jintao Huang, Chenglei Si, Ao Qu, Xiangru Tang, et al. The last human-written paper: Agent-native research artifacts. *arXiv preprint arXiv:2604.24658*, 2026.
- [12] Lennart Luettgau, Harry Coppock, Magda Dubois, Christopher Summerfield, and Cozmin Ududec. HiBayES: A hierarchical Bayesian modeling framework for AI evaluation statistics. *arXiv preprint arXiv:2505.05602*, 2025.
- [13] A. Keller, K. Kwegyir-Aggrey, R. Steed, A. Rao, J. Sharp, and A. Bergman. Expanding the AI evaluation toolbox with statistical models. NIST Trustworthy and Responsible AI NIST AI 800-3, National Institute of Standards and Technology (NIST), Gaithersburg, MD, 2026.
- [14] Yunbei Zhang, Janet Wang, Yingqiang Ge, Weijie Xu, Jihun Hamm, and Chandan K. Reddy. Stop comparing LLM agents without disclosing the harness, 2026. URL <https://arxiv.org/abs/2605.23950>.
- [15] Yilun Yao, Xinyu Tan, Chao-Hsuan Liu, Yaoming Li, Zhengyang Wang, Wenhan Yu, Zhewen Tan, Yuxuan Tian, Guangxiang Zhao, Lin Sun, Xiangzheng Zhang, and Tong Yang. Harness-Bench: Measuring harness effects across models in realistic agent workflows, 2026. URL <https://arxiv.org/abs/2605.27922>.
- [16] Lee J. Cronbach and Paul E. Meehl. Construct validity in psychological tests. *Psychological Bulletin*, 52(4): 281–302, 1955. doi: 10.1037/h0040957.
- [17] UK AI Security Institute. Inspect AI: A framework for large language model evaluations. <https://inspect.aisi.org.uk/>, 2026. Accessed 2026.
- [18] Meridian Labs. Inspect Scout: A transcript-analysis library. https://meridianlabs-ai.github.io/inspect_scout/, 2026. Accessed 2026.
- [19] Transluce. Docent: A behaviour-analysis platform for AI agents. <https://docs.transluce.org/introduction>, 2026. Accessed 2026-07-01.
- [20] METR. Brief independent investigation of agents’ behavior, reasoning and collaboration in the OpenAI / Hugging Face hacking incident. <https://metr.org/blog/2026-08-26-openai-hugging-face-incident-investigation/>, August 2026. 26 August. Contributors: Ryan Greenblatt, Ajeya Cotra, and Hjalmar Wijk.
- [21] Andrew Gelman and Eric Loken. The statistical crisis in science. In *The Best Writing on Mathematics 2015*, pages 305–318. Princeton University Press, 2016. doi: 10.1515/9781400873371-028.

- [22] Sara Steegen, Francis Tuerlinckx, Andrew Gelman, and Wolf Vanpaemel. Increasing transparency through a multiverse analysis. *Perspectives on Psychological Science*, 11(5):702–712, 2016.
- [23] Raphael Silberzahn, Eric L Uhlmann, Daniel P Martin, Pasquale Anselmi, Frederik Aust, Eli Awtrey, Štěpán Bahník, Feng Bai, Colin Bannard, Evelina Bonnier, et al. Many analysts, one data set: Making transparent how variations in analytic choices affect results. *Advances in Methods and Practices in Psychological Science*, 1(3): 337–356, 2018.
- [24] Anna Bavaresco, Raffaella Bernardi, Leonardo Bertolazzi, Desmond Elliott, Raquel Fernández, Albert Gatt, Esam Ghaleb, Mario Giulianelli, Michael Hanna, Alexander Koller, et al. LLMs instead of human judges? A large scale empirical study across 20 NLP evaluation tasks. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 238–255, 2025.
- [25] Justin D. Norman, Michael U. Rivera, and D. Alex Hughes. Reliability without validity: A systematic, large-scale evaluation of LLM-as-a-judge models across agreement, consistency, and bias. *arXiv preprint arXiv:2606.19544*, 2026. Submitted 17 Jun 2026.
- [26] OpenClaw Contributors. OpenClaw: Your assistant, on your devices, in your chats. GitHub repository, 2026. URL <https://github.com/openclaw/openclaw>. MIT licence. Accessed 2026-08-18.
- [27] A. M. Hanea, M. F. McBride, M. A. Burgman, B. C. Wintle, F. Fidler, L. Flander, C. R. Twardy, B. Manning, and S. Mascaro. Investigate discuss estimate aggregate for structured expert judgement. *International Journal of Forecasting*, 33(1):267–279, 2017.
- [28] Anthony O’Hagan, Caitlin E. Buck, Alireza Daneshkhan, J. Richard Eiser, Paul H. Garthwaite, David J. Jenkinson, Jeremy E. Oakley, and Tim Rakow. *Uncertain Judgements: Eliciting Experts’ Probabilities*. Statistics in Practice. John Wiley & Sons, 2006. ISBN 978-0-470-02999-2. doi: 10.1002/0470033312.
- [29] Magda Dubois, Ekin Zorer, Maia Hamin, Joe Skinner, Alexandra Souly, Jerome Wynne, Harry Coppock, Lucas Sato, Sayash Kapoor, et al. Seven simple steps for log analysis in AI systems. *arXiv preprint arXiv:2604.09563*, 2026.
- [30] Transluce. Analysis plans. <https://docs.transluce.org/analysis/analysis-plans>, 2026. Accessed 2026-09-23.
- [31] Toby D. Pilditch. Knowing when to stop: Bayesian optimal stopping for LLM evaluations, 2026. URL <https://arxiv.org/abs/2608.14425>.
- [32] Hjalmar Wijk, Tao Lin, Joel Becker, Sami Jawhar, Neev Parikh, Thomas Broadley, Lawrence Chan, Michael Chen, Josh Clymer, Jai Dhyani, Elena Elicheva, Katharyn Garcia, Brian Goodrich, Nikola Jurkovic, Holden Karnofsky, Megan Kinniment, Aron Lajko, Seraphina Nix, Lucas Sato, William Saunders, Maksym Taran, Ben West, and Elizabeth Barnes. RE-Bench: Evaluating frontier AI R&D capabilities of language model agents against human experts. *arXiv preprint arXiv:2411.15114*, 2024. METR. Also published at ICML 2025, PMLR 267:66772–66832.
- [33] Severin Field, Raymond Douglas, and David Krueger. AI researchers’ views on automating AI R&D and intelligence explosions, 2026. URL <https://arxiv.org/abs/2603.03338>.
- [34] The Anthropic Institute. When AI builds itself. Anthropic, June 2026. URL <https://www.anthropic.com/institute/recursive-self-improvement>. Accessed 2026-08-18.
- [35] Mingguang Chen, Licheng Wang, and Bo Qu. Recursive self-improvement in AI: From bounded self-refinement to autonomous research loops, 2026. URL <https://arxiv.org/abs/2607.07663>.
- [36] OpenAI. Finite time blowup for Navier–Stokes. Technical report, OpenAI, 2026. URL <https://cdn.openai.com/pdf/32d9f210-8b73-45e0-91bc-82a30aef8a9a/navier-stokes.pdf>.
- [37] Aleksandr Bowkis, Marie Davidsen Buhl, Jacob Pfau, and Geoffrey Irving. Automated alignment is harder than you think. *arXiv preprint arXiv:2605.06390*, 2026.
- [38] Paul Feyerabend. *Against method: Outline of an anarchistic theory of knowledge*. New Left Books, 1975.
- [39] Larry Laudan. The demise of the demarcation problem. In *Physics, philosophy and psychoanalysis: Essays in honour of Adolf Grünbaum*, pages 111–127. Springer, 1983.
- [40] Michael Polanyi. *Personal knowledge: Towards a post-critical philosophy*. University of Chicago Press, 1958.
- [41] Hui Chen, Miao Xiong, Yujie Lu, Wei Han, Ailin Deng, Yufei He, Jiaying Wu, Yibo Li, Yue Liu, and Bryan Hooi. MLR-Bench: Evaluating AI agents on open-ended machine learning research. In *Advances in Neural Information Processing Systems 38 (NeurIPS 2025), Datasets and Benchmarks Track*, 2025. doi: 10.48550/arXiv.2505.19955. URL <https://arxiv.org/abs/2505.19955>.

- [42] Mohammad Samar Ansari. Compound deception in elite peer review: A failure mode taxonomy of 100 fabricated citations at NeurIPS 2025, 2026. URL <https://arxiv.org/abs/2602.05930>.
- [43] Amanda Bienz, Carl Pearson, and Simon Garcia de Gonzalo. The case of the mysterious citations, 2026. URL <https://arxiv.org/abs/2602.05867>.
- [44] Peter Kirgis, Sayash Kapoor, Andrew Schwartz, Stephan Rabanser, David Africa, Konstantinos Voudouris, Viet Nguyen, Toby Pilditch, Magda Dubois, Harry Coppock, et al. Can AI agents conduct open-ended AI research? Early evidence from two case studies. *arXiv preprint arXiv:2607.27191*, 2026.
- [45] Anthropic. Claude Opus 4.7 system card. Anthropic, April 2026. URL <https://www.anthropic.com/claude-opus-4-7-system-card>. Published 16 April 2026. PDF: <https://www-cdn.anthropic.com/037f06850df7f8e871e206dad004c3db5fd50340/Claude%20Opus%204.7%20System%20Card.pdf>. Accessed 2026-08-18.
- [46] Jason Priem, Heather Piwowar, and Richard Orr. OpenAlex: A fully-open index of scholarly works, authors, venues, institutions, and concepts. *arXiv preprint arXiv:2205.01833*, 2022.
- [47] Crossref. Crossref documentation. Crossref, 2026. URL <https://www.crossref.org/documentation/>. Accessed 2026-08-18.
- [48] arXiv. arXiv.org e-print archive. arXiv, 2026. URL <https://arxiv.org/>. Accessed 2026-08-18.
- [49] Giuseppe Romeo and Daniela Conti. Exploring automation bias in human–AI collaboration: a review and implications for explainable AI. *AI & Society*, 41(1):259–278, 2026.
- [50] Meridian Labs. Meridian Labs, 2026. URL <https://meridianlabs.ai/>.
- [51] Chenglei Si, Diyi Yang, and Tatsunori Hashimoto. Can LLMs generate novel research ideas? A large-scale human study with 100+ NLP researchers. In *International Conference on Learning Representations*, volume 2025, pages 94003–94092, 2025.
- [52] Herbert A Simon. The scientist as problem solver. In David Klahr and Kenneth Kotovsky, editors, *Complex Information Processing: The Impact of Herbert A. Simon*, pages 375–398. L. Erlbaum Associates, 1989.
- [53] Charles Sanders Peirce. *Collected papers of Charles Sanders Peirce*, volume 5. Harvard University Press, 1934.
- [54] Michael D Skarlinski, Sam Cox, Jon M Laurent, James D Braza, Michaela Hinks, Michael J Hammerling, Manvitha Ponnappati, Samuel G Rodrigues, and Andrew D White. Language agents achieve superhuman synthesis of scientific knowledge. *arXiv preprint arXiv:2409.13740*, 2024.
- [55] Ronald Aylmer Fisher. *The design of experiments*. Oliver and Boyd, Edinburgh, 8th edition, 1966.
- [56] Thomas C Chamberlin. The method of multiple working hypotheses. *Science*, 15(366):92–96, 1890.
- [57] Deborah G Mayo. *Error and the growth of experimental knowledge*. University of Chicago Press, 1996.
- [58] John R Platt. Strong inference: Certain systematic methods of scientific thinking may produce much more rapid progress than others. *Science*, 146(3642):347–353, 1964.
- [59] Jun Shern Chan, Neil Chowdhury, Oliver Jaffe, James Aung, Dane Sherburn, Evan Mays, Giulio Starace, Kevin Liu, Leon Maksin, Tejal Patwardhan, et al. MLE-Bench: Evaluating machine learning agents on machine learning engineering. In *International Conference on Learning Representations*, volume 2025, pages 50466–50494, 2025.
- [60] Giulio Starace, Oliver Jaffe, Dane Sherburn, James Aung, Jun Shern Chan, Leon Maksin, Rachel Dias, Evan Mays, Benjamin Kinsella, Wyatt Thompson, Johannes Heidecke, Amelia Glaese, and Tejal Patwardhan. PaperBench: Evaluating AI’s ability to replicate AI research. *arXiv preprint arXiv:2504.01848*, 2025. OpenAI.
- [61] Runchu Tian, Yining Ye, Yujia Qin, Xin Cong, Yankai Lin, Yinxu Pan, Yesai Wu, Hui Haotian, Liu Weichuan, Zhiyuan Liu, et al. DebugBench: Evaluating debugging capability of large language models. In *Findings of the Association for Computational Linguistics: ACL 2024*, pages 4173–4198, 2024.
- [62] Clark A Chinn and William F Brewer. The role of anomalous data in knowledge acquisition: A theoretical framework and implications for science instruction. *Review of Educational Research*, 63(1):1–49, 1993.
- [63] Deepak Kulkarni and Herbert A Simon. The processes of scientific discovery: The strategy of experimentation. *Cognitive Science*, 12(2):139–175, 1988.
- [64] Kevin Niall Dunbar. How scientists really reason: Scientific reasoning in real-world laboratories. In *The Nature of Insight*, pages 365–396. The MIT Press, 1994. doi: 10.7551/mitpress/4879.003.0017.

A The Transect Package

The descriptions below apply to Transect package version 0.2.0. This appendix describes how to run an analysis (Section A.1), what the scanners produce (Section A.2), and how to read and reuse the outputs (Section A.3). It then covers family configuration (Section A.4), judging and auditing (Section A.5), extensions (Section A.6), and reproducibility and scaling (Section A.7).

A.1 Running an analysis

To analyse a run, the user supplies transcript logs and a **Spec**, the reusable configuration for an evaluation family. The Spec defines the behavioural vocabularies and task context. Analysis settings, including the judge models and number of repeated calls, are supplied separately to `transect()`. This function analyses the selected sample and epochs, generates the report, and returns a `TransectResults` object containing the dataframes.

`load()` rebuilds dataframes from a stored scan without model calls. `render()` generates a report from those results and available source transcripts (Section A.3).

Transect builds on Inspect Scout’s transcript representation, scanner abstraction and stored scan results [18].

Table 1 summarises the pipeline’s inputs and outputs.

Stage	Input	Output
Ingest	Logs in a supported format	Scout transcripts representing agent and sub-agent activity recoverable from the source
Select	Transcripts and sample/epoch settings	The selected sample and one or more epochs
Scan	Selected transcripts, Spec and analysis settings	Results from structural scanners and any enabled judged scanners, saved in a Scout scan store
Store	Stored scanner results	Tables of activities, recorded events, token use and audit information
Render	Dataframes and source transcripts	HTML reports organised by epoch; tables users can analyse or export

Table 1: Inputs and outputs of the five pipeline stages. Structural scanners extract or derive information without model calls; judged scanners call language models. Regenerating a report with excerpts requires access to the source transcripts.

Ingest. Ingestion presents supported evaluation logs as Scout transcripts so the same scanners can analyse them. Transect reads Inspect `.eval` logs and imports OpenClaw JSONL exports produced by telemetry-hal. The OpenClaw importer removes duplicate records from cumulative exports and reconstructs recorded messages, tool activity and sub-agent activity. What reviewers can inspect depends on the source records, including the available event and token-usage details. Additional formats require an adapter to this representation (Figure 1).

Select. A sample is an evaluation task or item; an epoch is an attempt at that sample. Each analysis selects one sample and one, several or all of its epochs before scanning. Inputs containing multiple samples require an explicit sample choice. If no epochs are specified, Transect selects the earliest epoch recorded as successful, or the earliest epoch if none is recorded as successful. This default favours successful attempts when available. Analyses requiring all attempts or a pre-specified sample of attempts should set the epoch selection explicitly.

When several epochs are selected together, the same configured scanners are applied to each attempt. Once scanning finishes, Transect builds the dataframes and generates separate reports for the selected epochs. The built-in dataframes retain results from all selected epochs; users can select an attempt by filtering the `epoch` column. Transcript identifiers link the results to their attempts, supporting separate review and comparisons across attempts.

Defaults. Table 2 lists selected defaults and text limits in the reviewed package. The Spec supplies vocabularies and context. These defaults have not been established as optimal settings.

The top-level controls are `judge_models`, `k_rolls`, `verify`, `verifier_model` and `verify_sample`. Phase and sub-agent spot-checking use different selection rules, described in Section A.5.

The phase scanner exposes `chunk`, `snippet_chars`, `verify_chunk` and `narrate`; the sub-agent scanner exposes `activity`. The task-prompt extraction helper accepts a `cap` argument; the built-in phase scanner uses its default. A

Parameter	Default or limit	Control
Judge models	None; built-in judged scanners disabled	Top level
Roll count k	1	Top level
Verifier	On for solo and k-roll; unavailable for cohort	Top level
Verifier model	First configured judge model	Top level
Verifier spot-check share	5%; default phase-sampling floor: 3 phases	Top level
Phase digests per classification chunk	40	Scanner
Phases per verifier chunk	8	Scanner
Phase narration	On	Scanner
Activity-informed sub-agent classification	Off	Scanner
Phase message excerpt, reasoning excerpt and each delegation string	300 characters each	Scanner
Extracted task-prompt context for phases	2,400 characters	Helper
Sub-agent task text	800 characters	Fixed
Phase fill confidence	0.3	Fixed
Review trigger: classifier confidence or agreement	Below 0.6	Fixed
Label-change gate: verifier confidence	At least 0.6	Fixed
Response caching	On; later rolls use separate cache scopes	See text

Table 2: Selected defaults in the reviewed package. Top-level controls are analysis-function arguments; scanner and helper controls require calls to the corresponding lower-level functions. Fixed values are set in the implementation. None of these controls is a built-in Spec field.

digest may contain several delegation strings, so the 300-character limit does not bound its total length. Response-cache controls are available on `decision_phases` and `cohort_llm_scanner`, but are not exposed by the top-level analysis function or the built-in sub-agent scanner (Section A.7).

This table is not intended as a complete record of the configuration needed to reproduce a run.

A.2 What the scanners produce

The turn axis. The report aligns behavioural phases, token use and recorded events on a shared **model-turn** axis. A turn is a model event with output; these events are numbered consecutively in normalised transcript order. Recorded sub-agent model turns use the same index, with separate lanes identifying the agents. Human-input markers use the same axis. An intervention recorded as a message is placed at the first model turn whose recorded input includes it; if that link is unavailable, its position is inferred from the recorded message and event order. Reconstruction may place a sub-agent’s activity after the event that spawned it. Turn positions alone do not establish elapsed time, exact ordering across agents or whether agents acted concurrently.

Scan: two kinds of scanner. A scanner extracts information or detects patterns in a transcript [18, 29]. Transect combines *structural* scanners, which extract recorded information without model calls, with *judged* scanners, which use language models to classify agent behaviour.

The built-in structural scanners are:

- `token_timeline` records the token usage reported for each model turn, its tool-call count and its agent lane. It also extracts sub-agent tool activity for placement on the timeline.
- `context_flush` extracts recorded context-compaction events and their available token counts before and after compaction. For Inspect logs, it also retains the summarisation call’s formatted prompt and the pre-compaction memory warning verbatim where recoverable. The report displays these texts under Eval setup / Core setup.
- `human_intervention` extracts mid-run messages marked as operator or human input, recorded answers to agent-initiated requests for input, and tool-approval decisions recorded by Inspect’s built-in human approver. Detection uses source annotations and event types.
- `eval_setup` extracts initial system and task prompts, agent and task settings, and evaluation-log metadata where available. These records describe the conditions under which the agent ran.

The dataframe builders use these records to calculate token measures. They can also infer compaction candidates from drops in context size, keeping inferred candidates distinguishable from recorded events (Section A.2).

The built-in *judged* scanners require models supplied through `judge_models` and matching vocabularies in the Spec. `decision_phases` uses the phase vocabulary; `subagent_classification` uses the sub-agent categories. Both use shared judging machinery with solo, repeated-call and multi-model modes. Their labels remain fallible model judgements. Voting, optional verification and the confidence or agreement information available in each mode are described in Section A.5.

decision_phases. The scanner divides the run into contiguous phases labelled from a closed vocabulary comprising the Spec’s categories and operational or fallback categories added where needed. An operational category covers setup and coordination; users can designate their own category for this purpose. The scanner adds `none_of_the_above` if it is absent from the Spec, allowing judges to flag activity that does not fit the supplied vocabulary. Providing this option is good classification practice: forcing a choice among inappropriate categories can produce consistent but incorrect labels. Judges can still overlook a mismatch. The vocabulary should be fixed before analysing runs intended for comparison (Section A.4).

Judges receive length-limited digests of main-agent message text, available recorded reasoning and delegation information, together with tool names. Reasoning excerpts are marked [THINKING]; a turn containing only readable recorded reasoning is also eligible. For redacted or summary-only reasoning blocks, the digest uses any readable summary supplied by the source. Recognised OpenClaw failure placeholders are excluded from the digests. The judges propose phase boundaries and labels for batches of digests, with the preceding phase label supplied as context where available. The results are combined and adjacent phases with the same label are merged, then mapped onto the shared turn axis. Some turns are assigned phase labels without their content being directly judged; the tables record the basis of each turn’s label (Section A.3). A continuous phase band therefore does not mean every turn was independently classified.

When enabled, narration runs after classification and any verification. The first configured judge model receives phase labels, a limited selection of each phase’s digests, and available task and Spec context. It produces a headline, short summary and titled groups of consecutive turns for the phase cards. If no usable narrative is returned for a phase, a label-based headline and empty summary are used instead.

Agreement on phase labels does not assess the accuracy of this generated prose. The classifications describe activities; conclusions about competence, correctness or safety require further assessment (Section A.5).

subagent_classification. This scanner assigns one label to each sub-agent span it classifies (a recorded sub-agent execution), using the Spec’s sub-agent categories and `none_of_the_above` for work that does not fit them. In the default *instruction-based* mode, the judge reads the delegation instructions and labels the work requested. If a sub-agent is asked to survey literature but spends its run debugging, the judge still assesses the request.

In the experimental *activity-informed* mode, the judge also receives a limited digest of recorded sub-agent activity, including message excerpts and tool-use counts where available. The prompt directs it to prioritise recorded work when this differs from the request. Both modes require task text recovered from delegation instructions or handoff messages. When activity records are absent, the activity-informed mode also relies on that text. Each classification covers the span as a whole, without dividing the sub-agent’s work into behavioural phases.

Token measures. `token_timeline` retains the reported input, output, total, cache-read, cache-write and reasoning token fields. Cache reads and writes count tokens retrieved from or stored in the prompt cache. The dataframe builder assumes Inspect’s ModelUsage convention: input tokens exclude cache reads and writes, and output tokens include reasoning tokens. Importers using other conventions must normalise their counts before dataframe construction. The OpenClaw importer uses a fixed mapping to these fields.

The package then calculates four measures for each turn:

- `context`: input plus cache writes and reads, representing the reconstructed input-context size.
- `new_work`: input and output, plus cache writes capped at the non-negative increase in context since the previous turn with usable token records in the same agent lane.
- `billable`: input, output, and all cache-write tokens. Cache reads are excluded.
- `turn_total`: input, output, cache writes and cache reads. This derived total can differ from the source’s reported `total_tokens`.

A missing input-token count or an all-zero input/output/cache record leaves these derived measures missing; missing components count as zero for this check. The check does not use reasoning or reported total tokens. Other missing components are also treated as zero when computing a derived value. The preceding context starts at zero in each lane and is preserved across skipped turns.

Interpreting token totals. The billable field is an unweighted token sum, not a monetary cost. It excludes cache reads even when a provider charges for them; prices can also differ for input, output and cache writes. Grouping token use by behavioural labels shows how tokens are distributed across the labelled activities; token expenditure alone does not measure scientific effort, research quality or efficiency.

A.3 Reading and reusing outputs

Report. In the default layout, the report opens with information about the transcript and evaluation setup, drawn from the available source records. Its timeline aligns available behavioural phases, token use, sub-agent activity and markers for human interventions and recorded or inferred context compactions. Reviewers can inspect these signals together and open phase cards containing labels, generated narratives and expandable transcript excerpts. Where the source provides readable reasoning or a reasoning summary, these excerpts display it in a muted [thinking] line above the turn’s text. The cards and excerpts can be explored within the generated report without Scout. Optional *Open in Scout Viewer* links support further exploration of the full transcript through a running Scout viewer. When generating a report, users can configure section order and add custom report elements based on their own scanners or analyses (Section A.6).

The reliability audit describes how the included judged classifications were produced and presents the available confidence, agreement and verification diagnostics (Section A.5). Reviewers can use these details to investigate warnings and assess the evidence supporting a classification. A final Scan execution & coverage section summarises execution across the stored scan, including all selected epochs. For each requested scanner, it reports completed transcripts against the recorded scope, or recorded scan attempts when scope information is unavailable. It also lists recorded analysis errors and available model usage from the scanners, grouped by scanner and model. Completion does not establish that usable labels were produced.

Dataframes. The returned `TransectResults` object contains pandas dataframes built from stored scanner results. These tables supply the report and can be queried, aggregated or exported for further analysis. Its `frames()` method returns the built-in tables by name. Custom tables are held separately in `layer_frames`, while optional `turn_tags` supports report filtering and grouping (Section A.6).

Rows and join keys. Each table has a defined row unit, such as a model turn, phase, sub-agent span or a judge’s record for a turn or span. Combine tables using transcript identity and the relevant turn, phase or span identifiers. Per-judge tables also distinguish the model and repetition number. When comparing separate analyses of the same transcript, retain the scan identity as well: the transcript identifier alone does not distinguish analysis configurations. A sample identifier alone is also insufficient because it can be shared across attempts. Before aggregating joined tables, check whether the join repeats records, for example when one turn has records from several judges or repetitions.

Table map. Table 3 summarises the returned tables, their row units and the fields used to link them. For built-in tables, combine the listed fields with `transcript_id` within an analysis. Custom layers define their own table schemas.

Missing data and label status. Interpret missing values using the field’s meaning and the scanner’s status. An absent token count differs from a recorded zero; missing judged output may mean the scanner was not run, a call failed or a judge refused to answer. A turn can have a phase label even when its content was not directly judged. Consult `basis` and `status` fields where available, alongside the scan records. Some derived token measures treat missing components as zero, so check their definitions before calculating totals or averages. An absent event record does not establish that the event did not occur.

Schema and provenance. The built-in tables carry a `schema_version` describing their table format. This field does not identify the package revision or complete analysis configuration. Reproducing an analysis also requires its input, configuration and software records (Section A.7).

Comparing and aggregating runs. Comparisons of activity or token allocation across runs require compatible category definitions and token-accounting conventions. Record differences in source coverage and judge settings, and distinguish deliberate experimental changes from differences that could confound the comparison. Phase indices locate phases within a transcript; they do not identify comparable activities across runs.

The research question and sampling design determine the statistical unit. Turns within a run and repeated judgements of those turns do not provide additional independent runs. Repeated attempts at the same task can also share task-specific influences. Depending on the question, an analysis may use run-level summaries, model dependence among observations, or both. A single transcript does not provide an empirical estimate of variation between transcripts. Specify the treatment of dependence and any multiple-comparison adjustment as part of the analysis design.

Table	One row represents	Linking fields within a transcript
transcript_info	Transcript identity and setup	No additional field
token_timeline, phase_turns	A model turn: token records or phase attribution	turn
flushes, interventions	A recorded or inferred compaction, or human-input event	turn; several events can share a turn
lane_activity	Sub-agent tool activity at a turn	turn, agent_span_id
phases	A behavioural phase	phase_index; turn_start, turn_end locate its range
turn_groups	A group of turns within a phase, with narration or fallback text	phase_index, group_index
phase_turn_votes	A judge member’s record for a turn included in the digests	turn, model, roll
subagents	A recorded sub-agent span	agent_span_id
subagent_votes	A judge member’s record for a sub-agent span	agent_span_id, model, roll
label_definitions	A category and its definition	surface, label
turn_tags (optional)	Custom tags associated with a turn	turn
layer_frames	Custom tables with layer-defined row units	Defined by the layer

Table 3: Table units and linking fields. `surface` identifies the phase, sub-agent or custom-layer vocabulary. A judge member is identified by its model and repetition number (`roll`). Linking fields locate related records; they are not necessarily unique row keys. `layer_frames` is a collection of custom tables, and `turn_tags` is optional; both are separate from the built-in tables returned by `frames()`.

Transect computes descriptive summaries and judge diagnostics to support human review. Judgements of agent performance remain with the human reviewer. Inference about capability differences across agents also requires evidence that the measures capture the intended capabilities, and a separate statistical analysis. Hierarchical modelling [12] and adaptive-stopping methods [31] address complementary questions about uncertainty and evaluation design.

Regenerating a report. Users can regenerate a report without rerunning the scanners or calling judge models. `load()` rebuilds the dataframes from a stored Scout scan, and `render()` generates the report. Supply the exact scan directory to select a particular analysis; when given a parent scans directory, `load()` selects the scan directory with the latest modification time. Custom layers must be supplied again through `extra_layers` so their stored results are included (Section A.6).

Rendering reads the source transcripts for excerpts and full sub-agent task prompts, and uses source tool-call counts when they are absent from the stored analysis. If these sources are unavailable, the stored analysis can still be rendered, but excerpts and any other content requiring a source read will be absent. Available excerpts are embedded in the report and can be expanded without a running Scout viewer. Optional full-transcript links require source access and a running Scout viewer, enabled through `viewer=True`. Keep the scan store and source transcripts to support later regeneration; exported analysis tables alone are not the input expected by `load()`.

A.4 Configuring an evaluation family

Family configuration defines which activities the analysis should distinguish and supplies the phase judges with relevant task context. Evaluation task descriptions, harness documentation and environment repositories provide a starting point.

The family Spec. The Spec holds a phase vocabulary, sub-agent categories and task context. Each category can include a description of the activity it covers. These definitions guide classification; they do not prescribe the order in which an agent must perform the activities. The built-in sub-agent classifier receives the category definitions but does not receive `Spec.context`. Users can also place custom analysis content in `Spec.extra`, which their custom layers must interpret (Section A.6).

Expert elicitation. Structured expert elicitation [27, 28] can help users address gaps in evaluation materials and review proposed category definitions. For example, experts can identify outcomes worth investigating and clarify the boundary between planning an experiment and carrying it out. Users can incorporate the resulting definitions and context into the Spec. Users can prepare questions and draft rubrics for review and revision with relevant experts.

Users should record which sources and expert contributions informed the rubric, alongside its validation status and supporting evidence (Section A.5). The package retains the vocabulary but does not maintain a dedicated record of expert contributions. Expert review can inform a rubric without establishing how accurately its classifications describe agent behaviour.

Analysis settings. Retain the Spec and invocation settings together to distinguish changes in behavioural definitions from changes in their application. Assessing a revised rubric or judging configuration requires a new analysis (Section A.7).

Developing the vocabulary. Development transcripts can help users identify activities to include in a preliminary vocabulary. Review these examples alongside evaluation materials and expert input, recording which sources informed each category and its definition. No fixed number of transcripts guarantees coverage: behaviours absent from the examples may still matter in later runs.

Keep the transcripts used to develop or revise the vocabulary separate from held-out data used to assess its performance or generality. A vocabulary can be explored on the examples that informed it, but those results do not provide an independent assessment. If findings from held-out transcripts inform a revision, treat those transcripts as development evidence and use data that did not inform the revision for an independent assessment. These are research-design practices; the package does not enforce the separation.

The built-in phase and sub-agent scanners include `none_of_the_above` even when it is absent from the Spec. Inspect these labels for possible gaps in the vocabulary. The fallback does not guarantee that every omitted behaviour will be detected. Custom scanners using `cohort_llm_scanner` classify against the supplied label list; the helper does not add a fallback category.

Case-study vocabularies. The CRUX case in Section 3 uses two vocabularies for custom judged analyses of orchestrator activity. The research-activity rubric focuses on general activities common to any AI R&D task. *Hypothesising* proposes a new mechanism, approach or explanation [51–53], while *Literature grounding* uses prior work to inform that direction [54]. *Experiment design* specifies what to vary and measure [55]; *Analysis design* chooses the statistical instrument for interpreting the evidence [21–23]. *Falsification* covers designing or executing a test intended to disconfirm the agent’s own hypothesis, with a failure condition set before the relevant results are seen [56–58]. Once work is specified, *Research engineering* implements, runs or monitors it [32, 59, 60], and *Debugging* addresses broken code [61]. *Evidence interpretation* connects an observed result to its scientific implications [62, 63], while *Reflecting* reassesses progress or strategy beyond a single result [64]. *Writing up* brings findings into the manuscript through drafting, revision and review, including figures, bibliography checks and compilation. A residual *Other* label covers generic operations, insufficient evidence, or work outside the vocabulary.

The paper-specific vocabulary focuses specifically on the TabPFN task’s concrete research questions and deliverables. *PFN problem framing* addresses harmful versus benign shift under the no-label deployment constraint; *PFN literature grounding* and *PFN solution ideation* cover positioning against prior detectors and developing ideas that exploit PFN affordances. Method development spans implementation (*PFN detector construction*) and theoretical guarantees (*PFN theory*); evaluation requires building the apparatus (*Benchmark construction*) and running detectors and baselines (*PFN model evaluation*). *Running ablations* isolates the source of a signal, while *Falsification tests* challenge the detector’s core claim against a named failure condition. Turning results into a deliverable brings together *Manuscript write-up*, *Self-review* against conference criteria, and *Code reproducibility* through runnable analyses. The vocabulary also tracks *Non-research-relevant engineering*, such as provisioning, permissions and file management, making operational work visible alongside these research activities. The case-study classification also includes a residual *Other* label for outputs outside these named categories.

A.5 Judging and auditing

The judging regime determines which consistency checks are available. A *solo* configuration uses one judge model for an initial judging pass over the classification inputs. A *k-roll* configuration repeats the same model, providing evidence about within-model stability. A *cohort* configuration uses several models, providing evidence about between-model agreement. The package keeps repeated calls to one model separate from multi-model voting: a cohort cannot also set `k_rolls` above one. Each member is identified by its model and repetition number.

Agreement can reflect shared errors, so neither repeated-call stability nor cross-model agreement establishes correctness. A solo judgement can carry stated confidence but supplies no repeated-judgement estimate of consistency. Verifier reviews and narrative generation can add further model calls. Cohorts can include models from different families. Model diversity does not guarantee independent errors or correct judgements.

Combining judgements. Voting selects the most frequent usable label for each unit. Phase classification combines votes at the judged-turn level before constructing phases; sub-agent classification combines votes for each classified span. If label counts tie, the tied label with the higher mean confidence wins, treating missing confidence as zero for this tie-break. A remaining tie is resolved by member order.

Vote agreement is the fraction of usable votes supporting the selected label. Refusals, failed calls, missing judgements and filled phase labels do not contribute votes, so agreement must be read alongside the number of usable voters and the recorded failures. One usable voter supplies no evidence of agreement between members. The available member records retain individual judgements and recorded failure states for inspection. Filled turns were included in the digests but inherit an earlier label when the returned phase segments leave them uncovered. Attributed turns lie outside the digests and are mapped onto the phase timeline.

The vote’s confidence is the mean of the available stated confidences among members supporting the selected label. It is distinct from vote agreement and is not a calibrated probability that the label is correct. Vote agreement is not chance-corrected. Where verification is enabled, a later review may change the final label. Distinguish the original voting results from the verifier’s subsequent assessment.

Selecting judgements for verifier review. The verifier adds a model review of selected classifications. It runs by default in solo and k -roll configurations; the package rejects verification with a multi-model cohort. Analysis settings can disable verification or select a verifier model; otherwise, the first configured judge model is used.

A phase is selected for review when the lowest confidence among its judged or filled digest turns is below 0.6, or the lowest available turn agreement is below 0.6. Review also covers phases containing at most two such digest turns between neighbouring phases that share a different label. For sub-agent classification, the triggers are stated confidence below 0.6 or vote agreement below 0.6 where available. These thresholds select candidates for review; they do not identify proven errors.

Both scanners also select additional judgements for spot checks. By default, the phase scanner targets the larger of three phases or 5% of all phases, rounded to a whole number, drawing from the remaining phases in label-wise rounds with a fixed seed. The target is capped by the number available. The sub-agent scanner uses a deterministic per-item draw with a default 5% selection probability, without a minimum count. `verify_sample` changes the spot-check setting; an explicit phase setting uses the requested fraction without the default minimum of three.

Verifier decisions and records. The verifier sees the earlier classification. For phases, it also receives selected turn digests and neighbouring phase labels; for sub-agent classifications, it reviews the classification input alongside the earlier answer and explanation. It proposes a label from the same vocabulary. A different label is applied only when the verifier’s stated confidence is at least 0.6. Phase changes are applied to the affected turn judgements, after which phases are reconstructed.

Available review records identify why a judgement was selected, the original and proposed labels, the verifier’s confidence and whether the change was applied. A review without a usable verdict leaves the earlier label unchanged. Sub-agent review records distinguish refusals and errors; phase verification counts phases without a usable verdict. After phase reconstruction, merged phases preserve all recorded reviews of their constituent original phases in `verifier_reviews`. A verifier that sees the original answer is not a blind independent assessment, even when it uses a different model. Agreement with the first judgement does not establish correctness, and an applied label change does not by itself establish improvement. A label-change rate measures model revisions, not independently established errors. Targeted selection and the different spot-check schemes also limit generalisation from reviewed cases to the run.

Reading the audit. For each included judged analysis, the reliability audit shows the judging regime, available confidence and agreement measures, verifier information, and breakdowns by label or judge where the required records are present. These views help reviewers distinguish disagreement about a label from missing judgements or changes introduced by verification.

Warnings can direct readers to the audit for more detailed diagnostics. For example, low agreement alongside poor member coverage warrants checking which judges returned usable labels before changing the rubric. A warning identifies a condition to investigate; its suggested explanations or follow-up actions do not establish the cause. Custom-layer audit participation is optional; Section A.6 describes the supported interface and recommended checks.

Agreement and member coverage. The package reports several agreement measures with different meanings. Per-unit vote agreement is the share of votes supporting the selected label, as defined above. Pooled pairwise agreement instead counts agreeing pairs of recorded member labels, divided by all within-unit pairs, pooling units with at least two non-missing labels. Units with more recorded labels contribute more pairs.

The package also computes nominal Krippendorff’s α and Gwet’s AC1, which compare observed agreement with different models of chance agreement. The AC1 calculation uses pooled pairwise agreement and category proportions, counting only categories observed in the eligible ratings. These are consistency measures, not classification accuracy. They use units with at least two recorded labels; the accompanying n counts the ratings in those units, not independent runs. The current implementation returns an unavailable coefficient when only one category is observed among eligible units, rather than treating that case as perfect chance-corrected agreement.

Member coverage reports, for each model and repetition, the fraction of its recorded unit rows that contain a judgement, alongside counts of recorded failures, missing responses and filled phase labels. Phase coverage counts directly judged rows; sub-agent coverage counts rows with an ok status. Coverage is conditional on the records present and cannot detect a member or unit omitted entirely from them. Read coverage alongside agreement: high agreement can coexist with low coverage.

Label-level summaries. The audit breaks down label counts, available confidence and agreement values, and the recorded source of each label by category. Phase summaries use directly judged turn rows; sub-agent summaries use spans with a recorded label. When the declared vocabulary is available, unused categories remain visible. An unused category means no included unit received that label; it does not establish that the corresponding behaviour was absent.

The minority-vote statistic asks how often a member’s use of a particular label differs from the selected label for the same unit. Its denominator is the recorded member votes for that label that can be matched to a decided unit; units whose final label came from the verifier are excluded. The provenance breakdown counts which recorded label source produced each decided label, such as a single judge, a vote or a verifier. These counts describe how decisions were made, not how many were correct.

Verifier-change summaries. For current built-in scanner outputs, the overall verifier-change rate is the fraction of completed reviews with usable verdicts that resulted in an applied label change. Per-label rates group these reviews by their original label; the spot-check rate uses completed reviews selected by the `random_sample` trigger. Phase summaries count reviews of the original phases, preserving them through later merging. Reviews without usable verdicts are counted separately and excluded from these rate denominators. These are summaries of the stored review records, not necessarily of every attempted review.

Read these rates with the selection and record-coverage limits above. A low change rate can reflect agreement with the initial labels or proposed changes below the confidence threshold for application; it does not establish a low error rate.

Intervals on diagnostic summaries. For counted proportions, including member coverage and verifier-change rates, the package uses nominal 95% Wilson intervals and returns no rate or interval when the denominator is zero. For audit summaries of mean confidence or agreement, it drops missing values and uses the normal approximation $\bar{x} \pm 1.96s/\sqrt{n}$, where s is the sample standard deviation and n is the number of retained values. These audit mean intervals are omitted below eight values.

These calculations do not account for dependence between turns, shared judge errors or the verifier’s selection process. Their nominal level therefore does not guarantee 95% coverage for a claim about a run or evaluation family. An interval around mean stated confidence describes those confidence values; it does not turn them into calibrated probabilities of correctness.

Scanner-level confidence summaries. Scanner outputs also contain confidence summaries calculated over different sets of values. A vote’s `confidence_pm` applies the normal-approximation half-width $1.96s/\sqrt{n}$ to the available stated confidences of members supporting its selected label. The helper returns zero when fewer than two values are available. This differs from the audit mean interval, which is omitted below eight values.

For phases, `confidence` averages the confidence values of the phase’s judged or filled digest turns, treating a missing value as zero. `confidence_spread` applies the same half-width helper to those turn values. This is a half-width for their mean, not their standard deviation or a measure of disagreement between judges. After an applied verifier change, affected turns receive the verifier’s confidence, and the resulting verifier-marked phase has its spread set to zero. A zero spread can therefore reflect insufficient values or assignment after verifier review, as well as identical confidence values; it is not evidence of certainty.

Warning thresholds. The current report uses fixed rules to flag diagnostics for inspection:

- Mean k -roll vote agreement below 0.80 or above 0.95 produces an amber warning.

- Cohort Krippendorff’s α below 0.66 produces a red warning; values from 0.66 up to, but not including, 0.80 produce an amber warning.
- Overall or per-label verifier-change rates of at least 0.20 produce an amber warning.
- Any applied change in the recorded spot-check subset produces a red warning.
- Where a judging regime is recorded, missing classifications produce an amber warning when their share is above zero but below 25%, and a red warning at 25% or more. For phases, this share counts `refusal`, `no_answer` and `missing_turn` rows among all non-attributed phase-turn rows; filled turns enter the denominator only. For sub-agent classification, it counts rows without a label among all recorded classification rows in the `sub-agents` table, including structurally recorded spans with no classification result.

These rules apply where the required metrics are available. The report also groups stated confidence into low (below 0.66), medium (0.66 to below 0.80) and high (at least 0.80) bands. These are display conventions, not calibrated levels of correctness.

Threshold crossings direct attention to the audit and source records. High self-agreement does not by itself diagnose label-name bias, and a spot-check label change does not prove that the original judgement was wrong. Conversely, the absence of a warning does not establish that the classifications are reliable or valid.

Evidence for interpreting classifications. Software tests can establish whether a scanner extracts or processes information as specified on the tested inputs. Repeated judgements and cross-model comparisons assess consistency. Neither establishes that a behavioural category captures the intended construct or that its labels are correctly applied to new transcripts.

External assessment compares classifications with an appropriate reference, such as independently collected expert annotations. Its interpretation depends on the reference’s quality, the assessment procedure and the transcripts represented. Expert review of category definitions is useful for developing a rubric, but is distinct from assessing how judges apply it. A comparison with another model provides additional model evidence; it does not become ground truth merely because that model is different or more capable.

Testing sensitivity to category names. The `scramble_spec` helper creates a copy of a `Spec` with neutral identifiers substituted for its phase and sub-agent labels. It returns a mapping for each classification type that `descramble` uses to translate output labels back to the original names. Descriptions, task context and operational-category flags are preserved. The helper does not rewrite label names mentioned inside those descriptions or the context, and it does not rename custom vocabularies in `Spec.extra`. Check which category names and definitions the judge actually receives, including reserved categories, before interpreting a comparison.

Users run the original and renamed configurations on the same inputs and judging settings, then compare outputs for the same turns or sub-agent spans after translating the labels back. Repeated unchanged runs help distinguish sensitivity to names from ordinary judging variation; cached responses cannot provide fresh evidence of that variation. Differences can motivate closer inspection of label wording or rubric definitions, but do not by themselves establish undue reliance on category names. Stability under renaming does not establish classification accuracy. Renamed runs are diagnostic artefacts and should be clearly identified as such; the package does not automatically mark their reports.

Ablations and other perturbations. An ablation removes a specified part of an analysis to test what depends on it. For example, removing an expert-supplied contextual instruction while retaining the transcript, category definitions and other analysis settings asks how much the classifications depend on that instruction. Before running the comparison, state what change is expected and which output will be compared. Keep other relevant conditions matched and compare the difference with variation under an unchanged configuration.

Check that the intended change reached the configuration or prompt actually used by the analysis. Compare outputs for matching source units, such as turns or sub-agent spans. Report failures and missing coverage in each condition alongside differences between conditions for matched units with usable outputs in both.

Removing a category also changes the available answers, so a resulting label change cannot be attributed solely to the removed category’s usefulness. Removing a report layer addresses a different question: its contribution to human review requires an appropriate reviewer study. Perturbations that disrupt transcript information or ordering likewise need an explicit prediction; preserving or losing a signal is not automatically the desirable outcome.

These are user-designed comparisons, supported by configurable inputs and stored outputs. The package does not provide a general ablation runner or automatically validate custom scanners through perturbation tests. Any claim that one configuration is more accurate needs an appropriate external assessment.

Scope of the case study. The case study illustrates how the combined report supports investigation of an agent’s research process. It does not provide a controlled estimate of reviewer time saved, findings missed or improvements in review decisions. Nor does it isolate the contribution of behavioural classification from that of making a long transcript navigable. Claims about classification accuracy or reviewer benefit require evidence tied to the relevant rubric, package version and evaluation setting.

Documenting validation evidence. Users should document what their categories are intended to represent, what assessment has been performed, the reference and data used, and the conditions to which the evidence applies. Distinguish expert review of category definitions from assessment of their application, and state when evidence is absent, preliminary or limited in scope. The package does not maintain a dedicated validation-status record; users must retain this documentation separately. Recording a status does not establish validity.

A.6 Extending the analysis

Custom scanner outputs can be converted into dataframes for further analysis or use in timeline layers and other report elements. Users can also model or analyse the dataframes generated by Transect and incorporate the resulting figures or other outputs as custom report elements. Users pass `Layer` objects through `extra_layers`. A layer can specify a scanner and a function that builds its dataframe, or accept an existing dataframe. It can also define report elements and phase-card tags. A custom report element therefore need not introduce another scanner. Custom tables are available through `results.layer_frames`, keyed by layer name. The returned dataframes also support separate dashboards, figures or summaries.

Custom scanners. Custom scanners can extract structural information or ask judge models to classify behaviour. For example, a structural scanner might identify recorded test failures, while a judged scanner might classify research activities using a family-specific rubric. Each scanner is registered with Scout’s `@scanner` decorator and supplied through a layer. Its results are saved in the Scout scan store and can be converted into a custom dataframe.

For classifications of individual items, `cohort_llm_scanner` provides shared judging support (Section A.5), including voting, optional verification and judge metadata. Tasks such as phase segmentation need additional logic to carry context between items; the built-in `decision_phases` scanner provides an implementation reference. Reusing the helper does not establish the validity of a custom rubric.

For custom classifications using `cohort_llm_scanner`, a loader can group several units into a window. With `batch=True`, the scanner judges that window in one initial call per member and retains separate results for each unit. Voting and any verifier review operate per unit. Results identify units with missing answers. For example, `reasoning_turns(batch=N)` with $N > 1$ groups up to N eligible main-agent turns. This loader’s digests contain available recorded reasoning, message text, tool names and delegation information. Turns containing only readable reasoning are eligible. The loader excludes recognised OpenClaw failure placeholders and turns with no text, readable reasoning or delegation information. Custom loaders can define different classification units and eligibility rules.

We recommend provenance and reliability review for every custom scanner. Record its inputs and method, the available evidence and any gaps. For structural scanners, document source coverage and extraction settings, distinguishing extracted, inferred and unavailable values; repeatability alone does not establish complete or correct extraction. The package offers an opt-in report audit for judged layers whose dataframes satisfy its audit contract. Other outputs require checks appropriate to their data and intended use; the package does not enforce universal audit coverage.

Custom report elements. A layer’s `section` defines how its results appear in the report. Built-in report blocks display numeric turn-by-turn charts, categorical bands, spans, event markers, sortable tables and explanatory text. For further customisation, `Component` calls a user-supplied function that returns an `Inspect Viz` component. These interfaces let users present figures or other outputs from their own modelling or analysis of Transect’s dataframes. Turn-based displays use the shared timeline; a table or other summary need not be organised by turn.

Blocks with dataframe inputs can read the layer’s dataframe or an explicitly supplied dataframe. If the dataframe includes `transcript_id`, rendering selects rows for the current transcript; otherwise it reuses the supplied data for each transcript. A custom `Component` function receives the current transcript’s context and must select any additional data it uses appropriately. Users should preserve transcript identity when an analysis is run-specific, so its outputs are not inadvertently repeated in other reports.

Custom report controls. A layer’s `tags` setting maps categorical columns in a per-turn dataframe to named tag groups. These tags appear on phase cards, provide filters for finding relevant episodes, and add grouping choices to the

token-use display. For example, a custom research-activity classification can let reviewers filter cards by activity and inspect token use under the same categories. Numeric analysis outputs can instead be shown in charts or tables.

Users set `section_order` when generating or regenerating a report. It places the listed sections in the requested order, then appends any remaining sections in their default order. Each transcript’s reliability audit follows its ordered sections, and the run-wide Scan execution & coverage section closes the report. This controls report layout without changing the underlying classifications. The optional `sensitivity` setting adds a user-supplied protective marking to the report title and fixed banners at the top and bottom. No marking is added by default.

Reading signals together. The shared timeline lets reviewers examine several aspects of the same episode. For example, they can inspect a context-compaction marker alongside a change in the activity label and a rise in token use, then return to the transcript to investigate what happened. The source records and scanner audit help reviewers assess how each signal was produced and what its available diagnostics support (Section A.5). A missing signal may reflect incomplete logging, a scanner that did not run, or a classification failure; it does not by itself show that the behaviour was absent.

Signals appearing near one another on the timeline do not establish a causal relationship. Nor do several outputs necessarily provide independent evidence: they may share transcript excerpts, category definitions, judge models or derived values. For example, token use grouped by phase depends on the phase labels and cannot independently verify those labels.

Related signals can still help reviewers identify patterns, inconsistencies and questions for follow-up. When drawing a conclusion from several signals, document their shared inputs and assumptions and assess what evidence each adds. The report does not automatically measure dependence or establish the validity of the conclusion.

A.7 Reproducing and scaling analyses

Records to retain. Retain the source transcripts, selected sample and epochs, family Spec, judging settings, custom-layer definitions and exact scan directory. Record the package revision and relevant dependency and provider-model versions where available, including local code changes. Together, these records distinguish changes in the evaluation evidence from changes in how it was processed.

Stored scanner results and dataframes carry useful provenance, including recorded judge identities, label definitions and decision sources. They do not replace the complete analysis configuration or a record of the software environment.

Stored results, cached responses and fresh judgements. Reloading a scan with `load()` rebuilds its dataframes from stored scanner results without calling judge models. A new `transect()` invocation performs another scan, but judge calls can reuse Inspect’s response cache when their cache keys match. Repeated calls within a k -roll configuration use separate roll-specific cache scopes, so a later run can reuse the response from each corresponding roll. A new scan directory alone does not ensure fresh judge responses.

For each repeat intended to assess judging variation, set `INSPECT_CACHE_DIR` to a new, empty directory before starting its analysis process. Verify that relevant calls did not replay cached responses. Caching remains active within each process. This setting controls Inspect’s model-response cache; custom scanners may have other reuse mechanisms.

Structural extraction makes no model calls, but its outputs can change with the source data, adapter or scanner implementation. Fresh judged outputs may vary even with unchanged settings; identical outputs are also possible and do not establish that a model or pipeline is deterministic.

Each `transect()` invocation on OpenClaw JSONL input imports the supplied files into a fresh transcript snapshot database under the scans directory. Duplicate transcript identifiers across the supplied files raise an error. Loading a stored scan does not repeat the import.

Comparing analysis revisions. Before comparing outputs, check that they refer to the same source transcripts and align the relevant sample, epoch, turn or span identifiers. Record changes to the vocabulary, input processing and judging settings. A change in category definitions or turn numbering can make a direct label comparison misleading.

When comparing configurations whose judged outputs can vary, estimate unchanged-configuration variation using fresh judgements under the relevant conditions. This provides context for a changed configuration; it does not supply a universal tolerance band or establish that a difference is an improvement. Comparisons intended to establish greater classification accuracy need an appropriate external assessment.

Documenting configuration choices. Document why a family uses particular categories, thresholds and judge settings, including their sources, assumptions and changes between analyses. Keep expert-supplied choices distinguishable from defaults and generated suggestions. The Spec and interface checks constrain the form of the configuration; they do not establish the scientific justification for its contents. Keep this documentation alongside the Spec and stored analysis, including the supporting evidence and its limits.

Assistance for users. The package ships skills for coding assistants that guide common tasks. `using-transect` covers Spec authoring, running an analysis, reading reports and analysing dataframes. `transect-diagnostics` guides investigation of disagreements, missing outputs and judging settings, including the label-renaming diagnostic. `add-a-layer` explains how to add scanners, report elements, tags and audit entries. `custom-ui` guides separate presentations of the results, including how to retain provenance, judging diagnostics and missing-data information. These skills provide instructions and examples; users remain responsible for selecting behavioural categories, reviewing proposed configurations and definitions, recording their sources and assumptions, and interpreting results.

Running `python -m transect.skills install` from the project directory copies the packaged skills into `.claude/skills` for Claude Code. Development-repository hooks for regression checks and other working practices are separate from this installer. They should not be assumed to run in a user’s package installation or another coding-assistant environment.

Cost and runtime. Judged-analysis cost depends on input text, the number of classification units, judging regime, verifier coverage, narration and response-cache reuse. Longer transcripts may require more phase-classification chunks; additional sub-agent spans or custom items may require more classifications. Repeated calls and cohort members add judging work, while cached responses can avoid provider requests. API charges depend on the chosen models’ input, output and cache prices. Tokens used for analysis are separate from the evaluated agent’s token totals.

Structural analysis requires computation, storage and report generation without calling judge models. Concurrency, batching and provider limits also affect runtime, which cannot be inferred directly from token counts or API charges. Custom-scanner batching groups initial classifications into shared calls (Section A.6); verifier reviews can add calls. No speed or cost advantage is established here.

Scaling oversight. Users can add scanners, relevant context or role-specific rubrics, or choose alternative judge models, to address further evaluation questions through the shared tables and report. These additions can also increase cost, disagreement and shared errors; more context or more judges need not improve classification quality.

Provenance and reliability audits help users inspect outputs, identify missing evidence and select findings for human review. Users can then assess whether further analysis is worth its cost and what confidence the evidence warrants. Demonstrating improved accuracy, reviewer performance or cost-effectiveness requires an appropriate comparison.