

FamilyBench: A Benchmark for Family-Coordination Extraction from Everyday Messages

The Sense Team
familybench.com

Draft — September 28, 2026

Abstract

Consumer AI increasingly acts as an *extraction agent*: it reads a natural, messy input — a forwarded school email, a coach’s group message, a photographed flyer — and turns it into the structured records a user depends on (calendar events, reminders, assignments to the right family member). This task resists existing evaluation paradigms: unlike code generation it has no executable oracle, unlike question answering its correctness is multi-field, and unlike either the correct output for the *same* input differs across users because it depends on private household state. We present **FamilyBench**, a benchmark for family-coordination extraction. Each instance pairs a frozen family *fixture* (members, children with grades and schools, timezone, and the current date) with an input message and hand-written per-field gold outcomes. A hermetic harness runs a production extraction pipeline in-process with persistence, search, and notification dependencies faked out — only the language-model calls remain live — and grades produced records against gold using a declarative vocabulary of named match types plus assertions on pipeline outcome state. Because extraction is stochastic, we adopt the reliability metric pass^k over n trials rather than single-shot accuracy. We describe the task, the dataset design, and the methodology, and evaluate three triage models (two providers, three capability tiers) at $n = 5$ trials. The evaluation separates the models under a drop-in prompt swap and exposes a *capability cliff* — a family-context filtering task the incumbent passes on every trial collapses to zero under a naive model swap — yet a neutral-prompt leaderboard shows that cliff is an artifact of prompt fit, not capability: the ranking flips. The study also reveals model-specific reliability defects (fields extracted non-deterministically) that single-shot evaluation hides. A prompt ablation isolates the value of the production system’s heavily tuned prompt — pass^{10} of 1.00 vs 0.40 for a generic baseline on tasks that exercise its rules, and no gap on tasks that need none. We release the harness, schema, and seed tasks as an extensible framework. This is a prototype-scale draft; Section 8 states the prerequisites for a full release.

1 Introduction

A growing class of consumer AI features are *extraction agents*. A user forwards an email or shares a photo, and the system produces the structured records that make the product useful: a calendar event with the right date, time, and location; a reminder assigned to the right person; a recipe; or nothing at all when the message is promotional noise. Sense, a family-organization assistant, is one such product: families forward the daily stream of school newsletters, activity schedules, and appointment reminders, and the system extracts a shared family calendar from them.

Evaluating this task is harder than it looks, and the difficulty differs from what existing benchmarks target:

- **No executable oracle.** Unlike SWE-bench [1], where a repository’s own unit tests decide correctness, there is no test suite for “did this email become the right calendar event.”
- **Multi-field, partial correctness.** An event has a title, date, start/end time, location, and a set of assigned family members. Getting the date right but the assignment wrong is a distinct, gradable failure that a single scalar answer (as in GAIA [7]) cannot capture.
- **Context-dependence on private state.** The *same* input has different correct outputs for different families. A newsletter naming four children should produce one event for the family whose child is Emma, and it should route a grade-3 field trip to the third grader, not the first grader. Correctness is a function of household state, not the message alone.
- **Stochasticity.** The pipeline is language-model-driven, so a single run under-measures: the question is not “can it ever get this right” but “how reliably.”

FamilyBench targets this task directly. Our contributions are:

1. A **task formulation** for family-coordination extraction — a frozen family fixture + an input message + per-field gold — analogous to SWE-bench’s (repository, issue, tests) but for structured record extraction (Section 3).
2. A **hermetic evaluation harness** that exercises the real production prompts and pipeline with persistence, search, and notification dependencies faked out, so that a change to a prompt is what moves the score (Section 4.1).
3. A **declarative grading vocabulary** — named per-field match types plus assertions on pipeline outcome state — following the data-not-code convention of WebArena [5] and BFCL [6] (Section 4.2).
4. Adoption of **reliability metrics** (pass^k , $\text{pass}@k$) over n trials for this stochastic task, following τ -bench [4] (Section 4.3).
5. A **multi-model evaluation** (three triage models, two providers) that separates systems on reliability, exposes a family-context capability cliff, and quantifies the cost of a drop-in model substitution (Sections 5–6), and a **prompt ablation** isolating the value of the production system’s tuned prompt (Section 7), plus a **system-agnostic runner** and release plan (Section 10) that let any model or system be scored against the identical tasks and grader.

2 Related Work

Code and terminal agents. SWE-bench [1] evaluates program repair on real GitHub issues, deciding correctness by running the repository’s own tests. Terminal-Bench [3] packages each task as a Docker environment with a hidden pytest suite. Both rely on an *executable* oracle. FamilyBench borrows their frozen-environment discipline and pass/fail-per-task reward, but its oracle is a set of declarative per-field assertions, because extraction correctness cannot be run as a test.

Tool-use and agentic benchmarks. τ -bench [4] evaluates customer-service agents against a database and a policy document, comparing final database state to a gold state and introducing the reliability metric pass^k over repeated trials. BFCL [6] scores function calls by AST or execution match against acceptable-value sets. WebArena [5] specifies grading declaratively via an evaluation block combining string, URL, and page-content checks. FamilyBench adopts τ -bench’s reliability metric and state-assertion idea, and WebArena/BFCL’s declarative named-match-type grading, specialized to multi-record extraction.

Knowledge and reasoning QA. GAIA [7] grades short factual answers by normalized exact match, stratified by difficulty level. FamilyBench similarly stratifies tasks by tier and normalizes before matching, but its outputs are structured records rather than a single answer string.

To our knowledge, no existing benchmark targets multi-field, family-contextual extraction from consumer messages, where the correct output depends jointly on a messy input and per-user state.

3 Dataset and Task Definition

3.1 Task formulation

A task instance is a tuple $(\mathcal{F}, m, \mathcal{G})$:

- $\mathcal{F}$, a **family fixture**: the owner, a set of members (adults and children, each with attributes such as grade and school), an IANA timezone, “today’s” date, and optionally pre-existing calendar events (*seed events*) the system must reconcile against.
- m , an **input message**: an email (subject, body, sender, send date) or an image attachment.
- $\mathcal{G}$, the **gold outcome**: the set of records (events, reminders) a competent human family-assistant would produce for *this* family from *this* message, specified field by field, plus optional assertions on the pipeline’s outcome counts.

A system under test is any function mapping $(\mathcal{F}, m)$ to a set of produced records and outcome counts. The gold withholds no part of the input; difficulty comes from the messiness of m and the context-dependence on $\mathcal{F}$, not from hidden information.

3.2 Difficulty tiers

Tasks are labeled by difficulty tier, in the spirit of GAIA’s levels and SWE-bench Verified’s difficulty field [2]:

- T1** a single clean event (one date, time, location, assignment);
- T2** a multi-event digest (a newsletter or schedule with several events);
- T3** a photo-only input (a recipe card, a flyer);
- T4** adversarial cases: filtering to the family’s own children, cross-message deduplication, non-event (promotional) messages that must produce nothing, and relative-date or timezone traps.

T4 tasks are where systems most differ; they encode failure modes observed in production triage.

3.3 Task manifest and schema

Each task is a single JSON manifest carrying metadata, the input, the fixture, and the evaluation block, following the single-record convention of WebArena/GAIA/BFCL; image attachments sit beside the manifest. A JSON Schema defines the format (Appendix A). Each manifest carries provenance (tier, category, tags, author, source) and a shared **canary** string so a leaked task is detectable in training corpora, as in Terminal-Bench and BFCL. An annotated example manifest appears in Appendix B.

3.4 Annotation protocol and de-identification

Following the SWE-bench Verified philosophy [2], gold is hand-written and small rather than auto-mined and large: a trusted, de-identified set is worth more than a noisy bulk one. The intended pipeline (Appendix C) takes a real forwarded message, strips personally identifying information, re-homes it onto a synthetic fixture family, and has an annotator write the expected outcome by

hand. A released benchmark requires multiple independent annotators and reported inter-annotator agreement; the seed tasks in this draft are single-annotator synthetic instances built to the same patterns, and we treat multi-annotator agreement as a release prerequisite (Section 8).

4 Experimental Setup

4.1 Hermetic harness

The harness invokes the production extraction pipeline *in-process*. The pipeline reads family state from its arguments and persists produced records through a database accessor and a search service, and it schedules notifications. The harness replaces exactly these boundaries with in-memory fakes: a recording database that also answers the candidate-lookup queries the deduplication path issues, stubbed search returning no candidates, and no-op notification scheduling. Family reads are served from the fixture. **The language-model calls are the only live dependency**, so the benchmark measures the real prompts and pipeline logic; a change to a triage or planning prompt is what moves the score. This mirrors SWE-bench applying the model’s patch to the real repository rather than a reimplementa-tion.

4.2 Per-field grading

Grading is declarative data, not per-task code. A gold record maps each field to a named match type:

Match type	Semantics
exact	normalized equality (dates, plain strings, counts)
time	equality after time normalization (16:00 $\equiv$ 4:00 PM)
fuzzy	token/semantic overlap (titles, locations)
set	set equality (member assignment)
present	field is non-empty

Strictness is deliberately per field: dates, times, and assignment are exact — these are what break for users — while titles and locations match fuzzily so harmless wording differences do not fail a task. Produced records are aligned to gold records greedily by date and title; an unmatched gold record is a recall miss and an unmatched produced record is a precision loss. Omitting a field leaves it ungraded; an explicit empty collection asserts “produce none.”

State assertions. Some correct behaviors are invisible in the produced records. When a forwarded message duplicates an event the family already has, “merged into the existing event” and “created a second copy” yield the same visible record but differ in the outcome count `events_created`. FamilyBench therefore supports assertions on pipeline outcome state, the analogue of τ -bench’s database-state comparison. A task’s reward is 1 iff all graded dimensions (events, reminders, state) pass and the run did not error, and 0 otherwise.

4.3 Reliability metrics

Because the pipeline is stochastic, we run each task n times and report the unbiased estimators of τ -bench [4] and Terminal-Bench [3]. For a task with c successes over n trials,

$$\text{pass}^k = \mathbb{E}_{\text{task}} \left[\frac{\binom{c}{k}}{\binom{n}{k}} \right], \quad \text{pass}@k = \mathbb{E}_{\text{task}} \left[1 - \frac{\binom{n-c}{k}}{\binom{n}{k}} \right]. \quad (1)$$

pass^k is the probability that all k i.i.d. trials succeed (a *reliability* measure), while $\text{pass}@k$ is the probability that at least one of k succeeds; $\text{pass}^1 = \text{pass}@1$ is the mean success rate. For a consumer product, pass^k at $k > 1$ is the honest metric: a family experiences the feature repeatedly, and a field that is right one run in three is not solved.

4.4 Artifacts

The harness maintains the predictions/results separation universal to these benchmarks: a *predictions* file (what the system produced, per trial) and a *results* file (harness-computed scores, pass^k , and explicit passed/flaky/failed identifier lists), joined by task identifier.

4.5 Evaluated systems

The production pipeline has two model-driven stages: a vision-capable *triage* model that extracts candidate items from the message, and a *planning* model that reconciles them against existing calendar state (deduplication, updates). We vary the triage model — the extraction “brain” — across three configurations while holding the planning model fixed (`claude-sonnet-5`), isolating the extraction stage: `gpt-5.6-luna` (the production default, OpenAI), `claude-sonnet-5` (Anthropic frontier), and `claude-haiku-4-5` (Anthropic small). We run each system on the 8 general seed tasks at $n = 5$ trials; the prompt ablation (Section 7) adds 5 further tasks authored to target specific tuned rules, for 13 in total.

A crucial caveat frames every number below: the triage *prompt* is tuned for the production model. Swapping the triage model without re-tuning the prompt therefore measures **the cost of a drop-in model substitution under a fixed production prompt** — an operationally important question (e.g. can a cheaper model replace the incumbent?) — not a prompt-controlled measurement of raw model capability. We report it as such.

5 Results

Table 1 reports per-task reliability ($c/5$ successes) and aggregate pass^k for the three systems. Three points stand out. First, there is **clear separation**: the production default reaches $\text{pass}^5 = 0.88$, while both substituted models sit at 0.75. Second, the separation is **task-localized, not uniform** — no system dominates every task; the substituted models each fail or flake on a *different* subset, so a single scalar would hide the structure that the per-task view exposes. Third, **several tasks are flaky rather than cleanly passed or failed** (e.g. `haiku-4.5` on the soccer task at 2/5, `gpt-5.6-luna` on the promotional task at 4/5), which only $n > 1$ trials reveal.

Model-fair leaderboard (reference prompt). The results above hold the model’s own tuned prompt fixed, so they answer an operational question, not a capability one. To rank the models fairly we instead hold a single neutral *reference* prompt fixed across all three, using the model-agnostic reference runner (Section 10) - the protocol standardized benchmarks use [7]. Table 2

Table 1: Main results: successes over $n = 5$ trials per task, and aggregate reliability. Triage model varied; planning fixed at **claude-sonnet-5**. Bold marks a task a system fails outright (0/5). Numbers reflect drop-in substitution under the production (OpenAI-tuned) triage prompt (Section 4.5), and 8 synthetic tasks — diagnostic, not a leaderboard.

Task	gpt-5.6-luna (OpenAI, default)	claude-sonnet-5 (Anthropic)	claude-haiku-4-5 (Anthropic, small)
t1_001 soccer practice	5/5	5/5	2/5
t1_002 dentist (rel. date)	5/5	5/5	5/5
t2_001 school newsletter	5/5	1/5	5/5
t2_002 sports schedule	5/5	5/5	5/5
t4_001 own-kids filter	5/5	0/5	0/5
t4_002 promo (no events)	4/5	5/5	5/5
t4_003 re-forward dedup	5/5	5/5	5/5
t4_004 two-kids field trips	5/5	5/5	5/5
pass ¹ (mean)	0.97	0.78	0.80
pass ⁵ (reliability)	0.88	0.75	0.75
pass@5 (any of 5)	1.00	0.88	0.88

Table 2: Model-fair leaderboard under the neutral reference prompt (single-shot reference runner, 8 general tasks, $n = 3$). Contrast own-kids with the tuned-prompt result of Table 1: the cliff vanishes and inverts.

	gpt-5.6-luna	claude-sonnet-5	claude-haiku-4-5
six non-adversarial tasks	3/3 each	3/3 each	3/3 each
t4_001 own-kids filter	1/3	1/3	3/3
t4_003 re-forward dedup	0/3	0/3	0/3
pass ³	0.75	0.75	0.88

shows the result on the 8 general tasks at $n = 3$, and it reverses the earlier picture. All three models solve the six non-adversarial tasks (3/3). They diverge only on the own-kids filter — where, crucially, the *capability cliff disappears*: under the OpenAI-tuned prompt this task was 5/5 for **gpt-5.6-luna** and 0/5 for both Anthropic models, but under the neutral prompt it is 1/3, 1/3, and 3/3, with **claude-haiku-4-5** strongest overall. The cliff was a **property of the prompt’s fit to the incumbent model, not of model capability**. (The re-forward dedup task is 0/3 for all three: the single-shot reference runner has no reconciliation stage, so no bare model can dedup — itself an honest measurement of what a full system adds.) This is the clearest argument in the paper for reporting model comparisons under a common, disclosed prompt, and for treating a product’s tuned prompt as a separate ablation.

Baselines and gameability. Two LLM-free baselines establish the floor and probe whether the tasks are shortcut-solvable: *empty* (produce nothing) and *heuristic* (a regex date extractor emitting one all-day event per date, no time, member, or reconciliation). Across the 13 tasks the empty baseline scores 0.08 (pass¹; it passes only the one “produce nothing” task, correctly by construction) and the heuristic scores 0.00 — far below any model’s 0.75–1.00. An early heuristic run exposed a gameable task (a buried-date task passable by “put an all-day event on the only

date present”); we added distractor dates so the naive extractor over-produces and fails precision while a real system must select the correct date. Reporting baselines is what turns “the tasks look hard” into “the tasks are not trivially solvable.”

Confidence intervals. All pass^k figures carry a percentile bootstrap CI over tasks (the unit of variation). At 13 tasks these intervals are wide — the honest signal that, at prototype scale, aggregate differences smaller than the interval should not be over-read. The runner reports them by default.

6 Analysis

A capability cliff on the own-kids filter. The starkest result is `t4_001`: the production model passes 5/5, while both substituted models score 0/5. Inspecting the predictions, the failure is *under-extraction*, not the over-extraction the task was designed to catch. Given a “Star of the Week” newsletter that names four children, only one of whom (Emma) is in the family, the production model extracts exactly one event assigned to Emma; the substituted models, under the same prompt, extract *nothing* — they drop the message entirely. The benchmark’s per-field grading distinguishes these failure modes: this is a recall failure (missed a real event), not a precision failure (fabricated events for the other children). Both are wrong, but they call for opposite fixes. This is also the clearest evidence for the drop-in-substitution caveat: a task the incumbent handles perfectly collapses to zero under a naive model swap, a concrete warning for anyone considering replacing the triage model to cut cost.

Reliability decay exposes defects one-shot accuracy hides. Several tasks are flaky, and the flakiness is model-specific. `haiku-4.5` extracts the soccer practice *time* on only 2 of 5 trials; `gpt-5.6-luna` hallucinates an event from the promotional email on 1 of 5 trials; the early-dismissal *time* on `t2_001` is extracted non-deterministically depending on the model. A single trial would have labeled each of these “solved” or “broken” at random. The pass^k curve — which decays from pass^1 to pass^5 exactly when a task is flaky — is what surfaces them, and the gap between pass^k and $\text{pass}@k$ (e.g. 0.75 vs 0.88 for the substituted models) quantifies how much apparent success is one-lucky-trial away from failure. For a feature a family relies on daily, reliability is the honest metric.

State assertions earn their keep. The deduplication result (`t4_003`, 5/5 for all systems) is only distinguishable from a silent failure through outcome state. Both “merged into existing” and “created a duplicate” leave one visible event record on the calendar; the produced-record view alone cannot tell them apart. The `events_created=0` assertion is what separates the intended behavior (reconcile) from the harmful one (duplicate), and the planning stage was observed choosing an update action against the seeded event on every trial. Record-level grading is necessary but not sufficient for extraction agents that also mutate existing state.

7 Prompt Ablation: Does the Tuning Help?

The production triage prompt is roughly 700 lines of hard-won domain rules. Is that tuning doing measurable work, or would a competent generic prompt do as well? We hold the model fixed (`gpt-5.6-luna`) and compare the production prompt against a *generic* baseline that preserves the JSON output contract (so grading is identical) but strips every domain rule — roster narrowing,

Table 3: Prompt ablation: successes over $n = 10$ trials, same model (gpt-5.6-luna), production prompt vs a generic baseline with the same output schema. The tuning’s value is concentrated on tasks that exercise a specific rule; tasks any prompt handles show no gap.

Task	Production	Generic	Rule the tuning supplies
recurring (biweekly)	10/10	0/10	recurrence schema; “every other” = interval 2
sent-home reminder	10/10	0/10	“sent home” → check-bag reminder
grade-roster picture day	10/10	4/10	narrow a per-grade roster to the family’s rows
spring break (multi-day)	10/10	10/10	(neither needs a special rule)
buried no-school date	10/10	10/10	(neither needs a special rule)
pass ¹⁰	1.00	0.40	

recurrence encoding, the sent-home reminder heuristic, and so on. The generic baseline is competent, not a strawman: it still instructs the model to use the family context and resolve relative dates, and it scores well on easy inputs.

Two observations shaped the experiment. First, on the original seed set — most of which any reasonable prompt handles — the two prompts differed by only ~ 0.12 pass⁵, a gap *smaller than the production prompt’s own run-to-run variance* (pass⁵ of 0.88 then 0.75 on identical configs). An easy task set cannot measure prompt quality: the signal washes into noise. Second, we therefore authored five tasks that each *require* a specific tuned rule, and ran $n = 10$ (Table 3).

The tuned prompt wins decisively where it matters: pass¹⁰ of 1.00 vs 0.40. The value is **concentrated, not diffuse**. On the three tasks that need a domain rule, the generic prompt fails outright (recurrence and sent-home: 0/10; it senses recurrence but has no schema to encode the interval, and never emits the sent-home reminder) or is unreliable (roster: 4/10). On the two tasks any prompt handles, the tuning adds exactly nothing. The methodological corollary is general: **a prompt (or any component) ablation only registers when the task set stresses what that component addresses**, and reporting reliability rather than one-shot accuracy is what keeps the 4/10 roster result from masquerading as either success or failure.

8 Limitations and Future Work

This is an honest prototype, not a finished benchmark. Before a full release the following are required:

- **Scale, real data, and independence.** Thirteen synthetic tasks are a framework demonstration. A credible benchmark needs on the order of hundreds of tasks curated from real, de-identified forwards spanning the production distribution. More pointedly, the current tasks are *not independent of the system under test*: they were authored with knowledge of the Sense pipeline, and several ablation tasks were constructed to exercise rules read directly from its prompt. That inflates the production system on exactly those tasks and is teaching-to-the-test. A credible version requires tasks authored by annotators blind to the pipeline, with the correct outcome decided from the message alone — the protocol we document but have not yet executed.
- **Multi-annotator gold and agreement.** Current gold is single-annotator. A release needs multiple independent annotators and reported inter-annotator agreement (e.g. Cohen’s κ on field-level correctness).
- **Prompt-controlled model comparison.** Our three-model study varies the model under

a fixed, production-tuned prompt, so it measures drop-in substitution cost rather than raw capability. A capability ranking requires re-tuning the prompt per model, and evaluating more than three configurations.

- **Trial budget.** Reliable pass^k estimates on borderline tasks require far more than $n = 3$ trials.
- **Photo (T3) coverage.** The image path is supported by the harness but not yet populated with tasks.
- **Fuzzy matching.** Token-overlap title matching may be too blunt for real titles; an LLM-judge variant is future work, as is a chat/tool-use extension grading the full interaction trajectory and final family-state.

9 Ethics and Responsible Use

FamilyBench concerns households and, unavoidably, children: fixtures include children’s names, grades, and schools, and inputs are the kinds of messages families actually receive. We take three precautions. First, **no real family data appears in released tasks**: real forwards are used only as private source material and are de-identified and re-homed onto synthetic fixtures before any gold is written; the seed tasks in this draft are fully synthetic. Second, **the harness is hermetic**: evaluation writes to an in-memory store and performs no network action except the model call under test, so running the benchmark cannot touch a real user’s calendar. Third, **data minimization**: fixtures contain only the attributes extraction actually consumes (name, grade, school, timezone), never contact details, precise locations, or health information beyond what a task explicitly requires. A released dataset should ship with a datasheet documenting sources, the de-identification procedure, consent basis for any real-derived material, and intended-use restrictions.

10 Release, Access, and Extensibility

FamilyBench is built to be run by others, not only by us. The task, the grading vocabulary, and the runner are separated from any one product’s code: a *System* is any function from (message, fixture) to produced records, and the grader scores that output. Two systems ship. The first is the full Sense pipeline, run hermetically. The second is a self-contained *reference runner* that calls a single model with the reference prompt (Section 7) and parses its JSON into records, touching none of the production extraction code. An external team ranks a new model by pointing the reference runner at it, or evaluates an entirely different system by implementing the one-method System interface; the tasks and grader are identical either way, so numbers are comparable. This is what lets the benchmark measure *the task* rather than our pipeline, and it cleanly separates a public leaderboard (models under the common reference prompt) from a system’s private prompt engineering (reported as the ablation of Section 7).

Release model. Because tasks embed family and children’s data (de-identified, but still), we do not plan a fully open dump. Following GAIA [7], we intend a small **public development split** — with gold, for iteration and for exercising the reference runner — and a larger **held-out test split** whose gold is withheld and scored through a submission endpoint. This both protects the underlying data and resists contamination; every task also carries the shared canary string. The datasheet described in Section 9 accompanies the release.

Acknowledgments

We thank the Sense users whose forwarded messages, in de-identified form, inform the task distribution, and colleagues who reviewed the evaluation methodology.

References

- [1] C. Jimenez, J. Yang, et al. SWE-bench: Can Language Models Resolve Real-World GitHub Issues? *ICLR*, 2024.
- [2] OpenAI. Introducing SWE-bench Verified. 2024.
- [3] The Terminal-Bench Team. Terminal-Bench: A Benchmark for AI Agents in Terminal Environments. 2025.
- [4] S. Yao, N. Shinn, et al. τ -bench: A Benchmark for Tool-Agent-User Interaction in Real-World Domains. 2024.
- [5] S. Zhou, F. Xu, et al. WebArena: A Realistic Web Environment for Building Autonomous Agents. *ICLR*, 2024.
- [6] F. Yan, H. Mao, et al. Berkeley Function Calling Leaderboard. 2024.
- [7] G. Mialon, C. Fourrier, et al. GAIA: A Benchmark for General AI Assistants. *ICLR*, 2024.

A Task Schema (summary)

A task manifest (`task.json`) has top-level keys `task_id`, `canary`, `tier`, `category`, `tags`, `author`, `source`, `notes`, `input`, `fixture`, and `evaluation`. `input` carries the message (`type`, `subject`, `body`, `sender_email`, `date`, optional `images`). `fixture` carries `owner_id`, `timezone`, `today`, `members` (`adults/children`), and optional `seed_events`. `evaluation` carries `events` and `reminders` (lists of `field→match-spec`) and an optional `state` block of count assertions. Each match spec is `{"match": <type>, "value": <expected>}`.

B Example Task (deduplication)

The re-forward deduplication task seeds one existing event and forwards the originating email again:

```
fixture.seed_events: [{id: existing_soccer_1, title: "Soccer practice",
  startDate: "2026-03-05", startTime: "16:00", ...}]
input: the same coach email that created it
evaluation.events: [{title: fuzzy "Soccer practice", startDate: exact "2026-03-05"}]
evaluation.state: {events_created: exact 0}
```

The `state` assertion is what distinguishes a correct merge from a duplicate; both leave one visible event.

C Annotation and De-identification Protocol

(1) Select a real forwarded message from the triage corpus. (2) Replace every personal identifier — family and child names, school names, email addresses, phone numbers, precise addresses — with synthetic values drawn from a fixed fixture family; preserve structure, dates (shifted consistently if needed), and phrasing. (3) Author the gold outcome field by field, recording only what a competent assistant would produce for that fixture family. (4) For a release, have a second annotator independently produce gold and compute field-level agreement; adjudicate disagreements. (5) Stamp the shared canary string.

D Seed Task Inventory

Task ID	Tier	Focus
t1_001_soccer_practice	T1	single event, time range, assignment
t1_002_dentist_relative_date	T1	relative-date resolution
t2_001_school_newsletter	T2	two events; dismissal time (flaky)
t2_002_sports_schedule	T2	three-game schedule
t4_001_own_kids_filter	T4	filter to the family’s own child
t4_002_promo_no_events	T4	non-event message; produce nothing
t4_003_reforward_dedup	T4	merge, not duplicate (state assertion)
t4_004_two_kids_field_trips	T4	grade-routed assignment
t2_003_recurring_biweekly	T2	recurrence encoding (ablation)
t2_004_spring_break_multiday	T2	multi-day collapse (ablation)
t4_005_grade_roster_picture_day	T4	per-grade roster narrowing (ablation)
t4_006_sent_home_reminder	T4	sent-home reminder heuristic (ablation)
t4_007_buried_noschool	T4	buried-event recall (ablation)