MedSkill: Progressive Skill Enrichment for Structured Clinical Information Extraction

Anonymous Author(s)

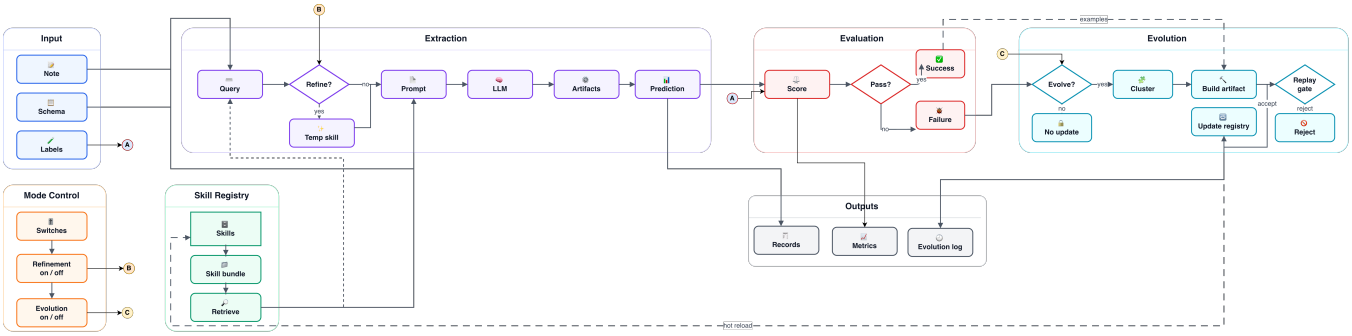

Figure 1: MedSkill architecture overview. A clinical note and task schema are processed through a schema-constrained extraction pipeline that retrieves relevant skills from a persistent skill registry. The mode-control switches determine whether the pipeline uses query-specific refinement, persistent evolution, or both. When refinement is enabled, MedSkill creates a temporary note-specific skill before prompting the LLM. The prediction is then post-processed through skill artifacts and scored against ground-truth labels, which are used only for evaluation and failure analysis. When evolution is enabled, failed records are clustered and converted into candidate skill artifacts. Candidate artifacts are accepted only if they pass a replay gate, after which the registry is updated and hot-reloaded for subsequent records. Solid arrows indicate the per-record extraction and evaluation flow, and dashed arrows indicate control or feedback paths.

Abstract

Clinical datasets increasingly support research, quality measurement, and decision support, yet much of the information needed for these workflows remains embedded in free-text notes. Clinical information extraction is therefore a high-stakes structured prediction problem. Systems must return valid fields and labels, apply clinical task rules, handle missing evidence, and avoid error patterns that can bias downstream analysis. We introduce *MedSkill*, a progressive skill enrichment framework for large-language-model based clinical extraction. MedSkill makes procedural knowledge explicit through three mechanisms. Schema grounding constrains outputs to task-defined fields and labels. Query-specific skill refinement creates temporary note-level guidance at inference time. Persistent skill evolution converts repeated extraction failures into reusable registry updates. We evaluated MedSkill with `gpt-4o` on two clinically distinct tasks, atrial fibrillation extraction from ECG reports ($n = 200$) and employment and housing extraction from social history notes ($n = 110$). Each task was tested under four ablation settings, schema grounding alone, schema grounding with refinement, schema grounding with evolution, and the combined refinement and evolution setting. Compared with the static ClinBench GPT-4o baseline, schema grounding alone improved F1 on both ECG (0.673 vs. 0.570) and SDOH (0.850 vs. 0.820). The strongest mechanism depended on the structure of the task. Query-specific refinement achieved the best ECG F1 (0.952), where extraction required note-level reasoning over negation, synonyms, and ambiguous rhythm language. Persistent evolution achieved the best SDOH trade-off (F1 = 0.896), where recurring social-history patterns could be learned as reusable rules. The combined setting did not always improve performance and regressed on ECG (F1 = 0.890), revealing that persistent skills can become harmful when the failure signal is sparse or biased. These results show that reliable clinical extraction requires more than a strong base model. MedSkill provides a practical path toward extraction systems that are schema-constrained, adaptive, inspectable, and able to learn from repeated failures while exposing when adaptation itself becomes unsafe.

CCS Concepts

• **Do Not Use This Code → Generate the Correct Terms for Your Paper**; *Generate the Correct Terms for Your Paper*; Generate the Correct Terms for Your Paper; Generate the Correct Terms for Your Paper.

Keywords

Do, Not, Use, This, Code, Put, the, Correct, Terms, for, Your, Paper

ACM Reference Format:
Anonymous Author(s). 2018. MedSkill: Progressive Skill Enrichment for Structured Clinical Information Extraction. In *Proceedings of Make sure to enter the correct conference title from your rights confirmation email*

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than the author(s) must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
Conference acronym 'XX, Woodstock, NY
© 2018 Copyright held by the owner/author(s). Publication rights licensed to ACM.
ACM ISBN 978-1-4503-XXXX-X/2018/06
https://doi.org/XXXXXXX.XXXXXXX

(Conference acronym 'XX). ACM, New York, NY, USA, 21 pages. https://doi.org/XXXXXXX.XXXXXXX

1 Introduction

Large language models (LLMs) are increasingly used as agents that can follow procedures, call tools, inspect external information, and revise intermediate outputs rather than only generate text. Early agent frameworks such as ReAct and Toolformer showed that integrating reasoning with actions or API calls can improve task performance, shifting progress from the model alone to the model embedded in a workflow [7, 12]. This shift has motivated recent work on agent skills, where procedural knowledge is packaged as reusable instructions, examples, schemas, scripts, and validation routines for a class of tasks. SkillsBench showed that curated skills can improve agent performance across task domains, with especially large gains in healthcare, while SkillNet and ClawKeeper highlight the broader role of skills in reuse, quality control, and safety governance [4–6]. Clinical information extraction is a natural setting for skill-based agents because valid outputs require more than general language understanding. A clinical extraction system must apply domain definitions, enforce label constraints, handle missing evidence, and verify that predictions follow the task specification. Prior medical LLM and agent studies have shown strong performance on clinical knowledge benchmarks and simulated clinical workflows, and ClinBench recently standardized structured clinical extraction across tasks including atrial fibrillation and social determinants of health [2, 8–11]. However, existing evaluations have mostly studied static prompting or fixed agent workflows. It remains unclear whether clinical extraction can improve through reusable, task-specific, and failure-driven skills.

We introduce MedSkill, a progressive skill enrichment framework for structured clinical information extraction from free-text clinical notes. MedSkill combines schema grounding, which constrains outputs to valid fields and labels, query-specific refinement, which creates temporary note-level extraction guidance, and persistent skill evolution, which converts repeated extraction failures into durable skill updates. We evaluate these mechanisms through an ablation design that isolates schema grounding alone, schema grounding with refinement, schema grounding with evolution, and the combined refinement and evolution setting. Experiments are conducted on two clinically distinct tasks, ECG atrial fibrillation diagnosis from MIMIC-IV-ECG and employment and housing extraction from MIMIC-III social history sections. Across these tasks, MedSkill is compared against ClinBench GPT-4o [11] as a baseline using the same model. The results show that the value of each mechanism depends on task structure. Query-specific refinement is most useful for ECG notes, where diagnoses are often implied through negation, synonyms, and ambiguous rhythm language. Persistent evolution is most useful for SDOH extraction, where recurring social-history patterns can be encoded as reusable rules. These findings suggest that clinical agent reliability depends not only on model capability, but also on how procedural knowledge is represented, validated, and updated over repeated extraction failures.

2 Methods

MedSkill was designed as a progressive skill enrichment framework for structured clinical information extraction from free-text clinical notes as shown in Figure 1. Given a clinical note and a task-specific output schema, the framework produces schema-constrained predictions for predefined fields and label values. The method combines three mechanisms: schema grounding, query-specific skill refinement, and persistent skill evolution.

Schema grounding provides the basic extraction constraint by limiting model outputs to the fields and labels defined by the task. Query-specific refinement adapts retrieved extraction guidance to the content of an individual note at inference time. Persistent skill evolution updates a durable skill registry when similar extraction failures recur across records.

2.1 Ablation Design and Operating Modes

The experimental design decomposed MedSkill into four operating modes defined by two independent configuration switches. The first switch, `use_refinement`, enabled query-specific refinement, which generates a temporary skill for each record. The second switch, `use_evolution`, enabled persistent skill evolution, which updates the registry after repeated extraction failures. The Cartesian product of these switches produced Schema Mode, Refinement Mode, Evolution Mode, and Full Mode, as summarized in Table 1.

All four modes used the shared extraction backbone described in Section 2.2. The modes differed only in whether refinement and evolution were enabled. This design made the ablation additive and directly comparable. Refinement Mode extended Schema Mode with one refinement call per record. Evolution Mode extended Schema Mode with failure-driven registry updates. Full Mode applied refinement using the current evolved registry state.

The ablation design supported three main comparisons. The difference between Schema Mode and Refinement Mode estimated the value of per-note adaptation. The difference between Schema Mode and Evolution Mode estimated the value of persistent cross-record learning. The comparison between the combined and single-mechanism modes tested whether refinement and evolution provided complementary benefits, interfered with one another, or behaved independently.

2.2 Shared Extraction Backbone

Before skill content is injected into the system prompt, MedSkill prepends an output constraints block derived directly from the task specification. This block specifies the exact field names and allowed label values. For optional fields, the model is instructed to omit fields that are not supported by explicit evidence. For required fields with a default label, such as `Unknown` in the SDOH task, the model is instructed to always output a value and to use the default label when evidence is absent. Detailed output-constraint examples are provided in Appendix A.1.

The retrieved skill body is appended under the `## Active Skills` section of the prompt. At the beginning of each run, the registry always contains the universal base skill `extract-general`. This skill is dataset-agnostic and instructs the model to read the output constraints first, extract only the listed fields, use only the valid label values, and omit optional fields when evidence is absent.

Table 1: Operating modes evaluated in the MedSkill ablation design.

Mode	use_refinement	use_evolution	Description
Schema Mode	No	No	Schema-grounded extraction from the universal base skill only.
Refinement Mode	Yes	No	Per-record ephemeral skill adaptation without persistent registry updates.
Evolution Mode	No	Yes	Persistent cross-record skill learning without per-record adaptation.
Full Mode	Yes	Yes	Query-specific refinement and persistent skill evolution active together.

For each record, `SkillRegistry.retrieve()` builds a retrieval query from the clinical note text and the target field constraints. Retrieval uses a hybrid BM25 and dense scorer:

$$\begin{aligned}\mathrm{Rel}(s,q) &= \lambda \cdot \mathrm{dense_norm}(s,q) \\ &\quad + (1-\lambda)\cdot \mathrm{bm25_norm}(s,q) \\ &\quad + \mathrm{overlap_boost}(s),\end{aligned} \tag{1}$$

where s is a candidate skill and q is the retrieval query. The default dense weight is $\lambda = 0.5$, and the default retrieval threshold is $\theta = 0.3$. When field constraints are present, λ is lowered to at most 0.35 to increase the influence of BM25 keyword matching. The retrieval threshold is also reduced by up to 0.12 to make retrieval more permissive, and an overlap boost of up to 0.25 is added when the skill text contains the target field names or allowed label values. Dense embeddings are computed with `sentence-transformers/all-MiniLM-L6-v2`. Only skills with scores above θ are returned. If no skill qualifies, extraction proceeds without injected skill content.

After retrieval, `Extractor._rerank_skills_for_constraints()` re-ranks the retrieved skills by constraint alignment. The alignment score counts matches between the skill text and the task field names or label strings. The system prompt and user message are then sent to `gpt-4o` with temperature 0 and JSON-object response formatting. After JSON parsing, MedSkill executes the active skill's deterministic artifact pipeline. The artifact pipeline is summarized in Appendix A.2.

2.3 Operating Modes

2.3.1 Schema Mode: schema-grounded extraction. Schema Mode evaluates the shared extraction pipeline with refinement and evolution disabled. It relies on the current registry and, at minimum, the universal `extract-general` skill described above. Its purpose is to enforce schema adherence, valid label values, and evidence-based extraction.

For a note x, Schema Mode retrieves up to k skills from the persistent registry $\mathcal{R}$ and predicts:

$$\hat{y}^{\mathrm{schema}} = f_\theta\big(x, s_{1:k}\big), \qquad s_{1:k} = \mathrm{Retrieve}\big(q(x);\mathcal{R}\big), \tag{2}$$

where $q(x)$ is the retrieval query built from the note and task constraints, $s_{1:k}$ are the retrieved skills, and f_θ is the LLM extractor. Schema Mode measures the floor performance of schema grounding alone.

2.3.2 Refinement Mode: query-specific ephemeral adaptation. Refinement Mode adds a query-specific refinement step before extraction. This step is implemented by `QuerySpecificRefiner`. It generates a single-use skill for the current note by synthesizing the retrieved registry skills through the lens of that note. The refined skill is saved to `refined_skills/{note_id}/SKILL.md` for observability, but it is not written to the persistent `skills/` registry.

For a note x, Refinement Mode first retrieves up to `top_k=3` skills using the same hybrid scorer as Schema Mode. It then synthesizes a refined skill s_r and extracts from that skill:

$$s_r = \mathrm{Refine}\big(x, s_{1:k}\big), \qquad \hat{y}^{\mathrm{refine}} = f_\theta\big(x, s_r\big). \tag{3}$$

The refinement call receives a structured prompt containing the skill creation instructions, the task specification, the output constraints, and the retrieved skills. The resulting skill may include note-specific inference rules, such as mapping "sinus tachycardia" to `NotAF`, handling negated atrial fibrillation mentions, or discarding irrelevant retrieved material. Extraction is then performed with `Extractor.extract()` using `override_skill_md=refined_md`, which bypasses registry retrieval for the extraction call. Refinement Mode measures the value of per-record adaptation on top of schema grounding.

2.3.3 Evolution Mode: persistent cross-record learning. Evolution Mode disables query-specific refinement and enables the failure-driven evolution engine. Each record is extracted using the current registry state. When a prediction fails evaluation, MedSkill creates a `FailureTrace` and adds it to the failure clustering pipeline. The evolution engine uses clusters of similar failures to propose, build, validate, and apply persistent changes to the skill registry.

For each failure, the trace encodes a semantic representation containing the task fields, artifact hint, error types, ground truth, the last 600 characters of the note, and the first 500 characters of the raw model response. This text is embedded with `all-MiniLM-L6-v2`. A failure trace with embedding e is assigned to an existing cluster if its cosine similarity to the cluster centroid μ exceeds the semantic threshold $\theta_c = 0.72$:

$$\mathrm{sim}(e,\mu) = \frac{e \cdot \mu}{\|e\|\|\mu\|}. \tag{4}$$

If no existing cluster satisfies the threshold, a new singleton cluster is created. Cluster centroids are updated as an online mean:

$$\mu_t = \frac{(t-1)\mu_{t-1} + e_t}{t}. \tag{5}$$

A cluster triggers the evolution pipeline when it reaches the configured batch size or when a smaller set of failures is highly coherent:

$$|C| \geq b \quad \text{or} \quad \left(|C| \geq \max\left(2, \left\lfloor \frac{b}{2} \right\rfloor\right) \text{ and } \overline{\mathrm{sim}}_C \geq \alpha\right), \tag{6}$$

where $b = 5$ is the evolution batch size and $\alpha = 0.88$ is the adaptive trigger threshold. The minimum size of two prevents single failures from triggering early evolution events.

Each triggered sub-batch passes through a strict evolution pipeline. The proposer selects whether to create or edit an artifact, the builder materializes the candidate, and the admission validator replays the candidate against the triggering failure sub-batch before any disk write occurs. The complete artifact-type table and target-normalization details are provided in Appendix **??**.

Let $\mathcal{B}$ denote the triggering failure sub-batch and let $B = |\mathcal{B}|$ be the number of failed records in that sub-batch. For each record $i \in \{1, \ldots, B\}$, r_i^{orig} denotes the reward obtained by the original registry state before applying the candidate artifact, and r_i^{replay} denotes the reward obtained when the same record is re-extracted with the candidate artifact temporarily overlaid. The admission validator computes the mean original reward, the mean candidate reward, and the candidate improvement as follows:

$$\bar{r}_{\text{base}} = \frac{1}{B}\sum_{i=1}^{B} r_i^{\text{orig}}, \qquad \bar{r}_{\text{cand}} = \frac{1}{B}\sum_{i=1}^{B} r_i^{\text{replay}}, \qquad \delta = \bar{r}_{\text{cand}} - \bar{r}_{\text{base}}. \tag{7}$$

In Equation 7, $\bar{r}_{\text{base}}$ is the average reward of the current system on the failed sub-batch, $\bar{r}_{\text{cand}}$ is the average reward after replaying the same records with the candidate artifact, and δ is the estimated improvement produced by the candidate. The candidate is admitted only if the estimated improvement exceeds the minimum admission threshold:

$$\delta \geq \Delta_{\min}, \qquad \Delta_{\min} = 0.02. \tag{8}$$

If $\delta < \Delta_{\min}$, the proposal outcome is recorded as rejected and no registry change is made. If the candidate is admitted, the skill manager decides whether to add, merge, or discard the candidate. Accepted artifacts are written to disk and the registry is hot-reloaded. After all evolution events have completed, MedSkill re-evaluates all records from scratch if any evolution event was accepted. The reported metrics for Evolution Mode are therefore based on the final skill state, not only on the online predictions made during the original record order.

2.3.4 Full Mode: refinement with persistent evolution. Full Mode activates both query-specific refinement and persistent evolution. The per-record extraction flow is the same as Refinement Mode. The difference is that failed predictions are also added to the evolution pipeline. In this mode, a failure trace records a post-refinement failure, meaning that the ephemeral refined skill was insufficient for the note.

For record i with registry state $\mathcal{S}_t$ at processing time t, Full Mode predicts:

$$\hat{y}_i^{\text{full}} = f_\theta\left(x_i, \text{Refine}\left(x_i, \text{Retrieve}(q(x_i); \mathcal{S}_t)\right)\right). \tag{9}$$

When a failure cluster C satisfies the trigger condition in Equation 6, the registry is updated:

$$\mathcal{S}_{t+1} = \text{Evolve}(\mathcal{S}_t, C). \tag{10}$$

The interaction between refinement and evolution occurs because future refinement calls retrieve from the evolved registry. An accepted evolution event changes the base skill state from which later note-specific skills are synthesized. As a result, patterns that refinement previously handled independently can become persistent registry knowledge, while refinement can focus on note-specific edge cases. Full Mode therefore measures the combined effect of ephemeral per-note adaptation and durable cross-record learning. The approximate LLM call structure for each operating mode is provided in Appendix A.4.

2.4 Datasets, Evaluation, and Experimental Configuration

The ECG task uses MIMIC-IV-ECG records [3]. The input consists of free-text ECG interpretation narratives generated by cardiologists from waveform analysis. The extraction task is binary classification with one output field:

$$\texttt{diagnosis} \in \{\texttt{AF}, \texttt{NotAF}\}. \tag{11}$$

The ECG benchmark contains 200 records and is class balanced, with 100 `AF` and 100 `NotAF` examples. The main difficulty is that the target label is often not stated literally. Correct extraction requires synonym inference, negation handling, and ambiguity resolution.

The SDOH task uses the MIMIC-SBDH dataset [1]. Only the social history section is used as input. The evaluated sample contains 110 records. The task extracts two fields simultaneously from each note:

$$\begin{aligned} \texttt{employment} &\in \{\texttt{Employed}, \texttt{Unemployed}, \texttt{Unknown}\}, \\ \texttt{housing} &\in \{\texttt{Housing}, \texttt{Homeless}, \texttt{Unknown}\}. \end{aligned} \tag{12}$$

Both SDOH fields are required. The label `Unknown` is used when the corresponding field is not explicitly mentioned in the note. For housing, `Unknown` is not inferred from discharge destination. Additional dataset definitions and label distributions are reported in Appendix B.

Each record is evaluated against its ground-truth field values. Let x be a clinical note, $y^* = \{f_1, \ldots, f_N\}$ be its ground-truth fields, and $\hat{y}$ be the model prediction. Each ground-truth field contributes equally to the reward. The per-record reward is:

$$r(\hat{y}, y^*) = \frac{1}{N}\sum_{i=1}^{N} \mathbb{1}\left[\hat{y}_{f_i} = y^*_{f_i}\right]. \tag{13}$$

A record passes only when its reward reaches the pass threshold τ:

$$\text{pass}(x) = \mathbb{1}\left[r(\hat{y}, y^*) \geq \tau\right], \qquad \tau = 1.0. \tag{14}$$

Thus, all ground-truth fields must be correct for a record to pass. Field matching is case-insensitive and permits substring tolerance. In addition to pass rate and reward, token-set micro-averaged precision, recall, and F1 are computed across all records. For each field, the predicted and ground-truth values are normalized to lowercase token sets. True positives, false positives, and false negatives are computed as set intersections and differences:

$$TP = |\hat{T} \cap T^*|, \qquad FP = |\hat{T} - T^*|, \qquad FN = |T^* - \hat{T}|. \tag{15}$$

Table 2: ECG atrial fibrillation diagnosis results, $n = 200$. ClinBench is an external GPT-4o static-prompt baseline. EM denotes exact match.

Condition	F1	P	R	EM	Failures
ClinBench GPT-4o	0.5700	–	–	–	–
Schema Mode	0.6731	0.9541	0.5200	0.5200	95/200
Refinement Mode	**0.9524**	0.9548	0.9500	0.9500	9/200
Evolution Mode	0.9450	0.9450	0.9450	0.9450	5/200
Full Mode	0.8895	0.9153	0.8650	0.8650	19/200

where $\hat{T}$ is the set of tokens in the model prediction and T^* is the set of tokens in the ground-truth value. The operators $\cap$ and $-$ denote set intersection and set difference, respectively.

Counts are accumulated across all fields and records. Micro-averaged precision, recall, and F1 are then computed as:

$$P = \frac{\sum TP}{\sum TP + \sum FP}, \qquad R = \frac{\sum TP}{\sum TP + \sum FN}, \qquad F1 = \frac{2PR}{P + R}. \tag{16}$$

All full-set experiments used `gpt-4o` with temperature 0. For each dataset, all operating modes were evaluated on the same record set. The Schema Mode run defined the sampled record identifiers, and the same identifiers were propagated to the remaining conditions. The full experimental configuration is summarized in Appendix B.3.

3 Results

ClinBench GPT-4o was used as the external comparison baseline but it does not use schema grounding, a skill library, query-specific refinement, or persistent skill evolution. The ClinBench baseline therefore represents a static prompting reference point for evaluating the added value of MedSkill's schema and learning mechanisms.

3.1 Overall Performance

Table 2 summarizes ECG performance across the four MedSkill modes and the ClinBench baseline. Schema Mode improved over ClinBench from 0.5700 to 0.6731 F1, but its recall remained low because many records returned no output. Refinement Mode achieved the best ECG performance, reaching 0.9524 F1. Evolution Mode produced a similar F1 score of 0.9450 after seven evolution events. Full Mode, which combined refinement and evolution, underperformed both single-mechanism adaptive modes with 0.8895 F1.

Table 3 summarizes SDOH performance. In contrast to ECG, refinement alone did not improve the SDOH overall F1 score. Schema Mode and Refinement Mode both achieved 0.8500 F1. Evolution was the dominant mechanism for SDOH, increasing F1 to 0.8955 and reducing failures from 30 to 20 records. Full Mode matched Evolution Mode in overall F1, but it required more evolution events and substantially more tokens, as shown in Appendix D.

Across datasets, the best ECG mode was Refinement Mode, whereas the best SDOH modes were Evolution Mode and Full Mode. Both tasks exceeded the target performance thresholds: the best ECG F1 was 0.9524, and the best SDOH F1 was 0.8955. The cost-efficiency frontier in Figure 2 shows that Evolution Mode provided the best cost-performance trade-off on both datasets. For ECG, Evolution Mode achieved nearly the same F1 as Refinement Mode at a lower token cost. For SDOH, Evolution Mode achieved the same overall F1 as Full Mode while using approximately one-third of the tokens per record.

Table 3: SDOH extraction results, $n = 110$. The task jointly evaluates `employment` and `housing`.

Condition	F1	Emp. F1	Housing F1	Failures
ClinBench GPT-4o	0.8200	0.8200	0.8200	–
Schema Mode	0.8500	0.8273	0.8727	30/110
Refinement Mode	0.8500	0.8182	0.8818	30/110
Evolution Mode	**0.8955**	0.8636	**0.9273**	20/110
Full Mode	**0.8955**	**0.8727**	0.9182	20/110

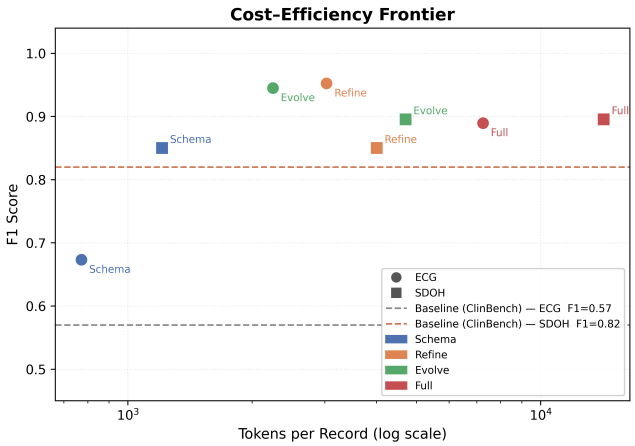

Figure 2: Cost-efficiency frontier for all MedSkill mode and dataset combinations. The x-axis shows tokens per record on a log scale, and the y-axis shows F1. Dashed horizontal lines indicate ClinBench GPT-4o baselines for ECG and SDOH.

3.2 ECG Atrial Fibrillation Diagnosis

The ECG task required binary extraction of `diagnosis` as either `AF` or `NotAF`. The strongest ECG result came from Refinement Mode, which improved F1 by 0.2793 over Schema Mode and by 0.3824 over ClinBench. This improvement reflects the importance of per-note reasoning for ECG narratives, where the target label is often implied by rhythm language, negation, or ambiguous diagnostic phrasing rather than stated literally.

Schema Mode had high precision but low recall. It achieved 0.9541 precision but only 0.5200 recall, producing 95 `pass=0` failures. The main cause was not wrong classification, but refusal. As shown in Table 4, 91 of the 95 Schema Mode failures were empty outputs. In these cases, the schema constraint prevented the model from producing unsupported labels when the note did not explicitly contain a valid label string. This behavior improved reliability relative to unstructured prompting, but it reduced recall.

Table 4: ECG failure decomposition by mode. Total failures are based on the recorded `pass=0` outcome. Wrong-answer counts are derived from direct prediction-versus-ground-truth comparison and can exceed `pass=0` failures when false negatives are not represented in the recorded failure count.

Mode	Fail	Empty	Wrong	FP	FN
Schema	95	91	5	4	1
Refinement	9	1	9	8	1
Evolution	5	0	11	5	6
Full	19	11	16	8	8

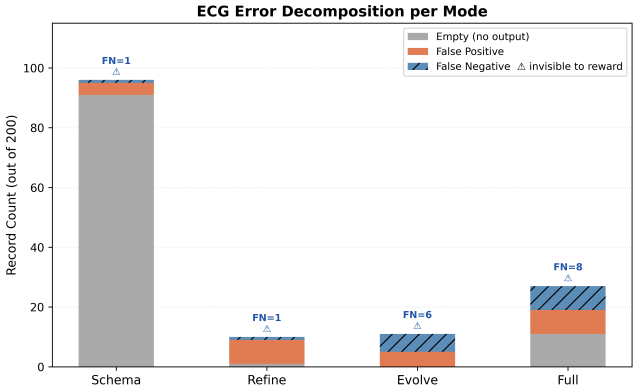

Figure 3: ECG error decomposition by mode. Bars separate empty outputs, false positives, and false negatives. False negatives are shown separately because direct output inspection identified AF records predicted as `NotAF` that were not represented in the recorded `pass=0` failure count.

Figure 3 visualizes the same failure structure. Schema Mode was dominated by empty outputs, whereas Refinement Mode converted most refusals into predictions. Evolution Mode eliminated empty outputs completely, but its false-negative segment increased. This pattern indicates that the evolved `extract-diagnosis` skill learned to reduce penalized false positives and refusals, while also introducing unpenalized false negatives.

A direct audit of ECG predictions revealed an important reward-signal asymmetry. The recorded `pass=0` failure count captured empty outputs and false positives, but AF records predicted as `NotAF` were not represented as failures in the evolution signal. This explains why Evolution Mode reduced the recorded failure count to five while still producing 11 wrong placements by direct comparison. The skill evolution process optimized the failure signal it received, and that signal favored suppressing false positives and refusals over correcting false negatives. Clinically, this direction is important because missed AF cases may be more consequential than false alarms. The ECG confusion matrices and cross-mode failure overlap are shown in Appendix C.

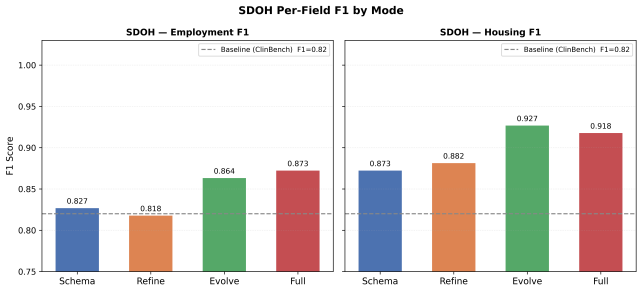

Figure 4: SDOH per-field F1 by operating mode. Dashed lines show the ClinBench GPT-4o baseline. Housing achieved higher F1 than Employment in every MedSkill mode, and both fields benefited from persistent evolution.

3.3 SDOH Employment and Housing Extraction

The SDOH task differed from ECG in two important ways. First, both output fields were required, and each field allowed `Unknown` as a valid default label. As a result, none of the SDOH modes produced empty outputs. Second, many relevant SDOH expressions were formulaic and recurring across records. This made the task better suited to persistent evolution than to per-note refinement.

The per-field F1 results in Figure 4 show that Housing was consistently easier than Employment. Housing F1 ranged from 0.8727 to 0.9273, whereas Employment F1 ranged from 0.8182 to 0.8727. Evolution improved both fields, but the gain was larger for Housing. Evolution Mode increased Housing F1 from 0.8727 to 0.9273 and Employment F1 from 0.8273 to 0.8636. Full Mode slightly improved Employment F1 relative to Evolution Mode, but it slightly reduced Housing F1.

The largest Employment error was `Unemployed`to `Unknown`. Schema Mode and Refinement Mode each produced eight such errors, whereas Evolution Mode and Full Mode reduced this count to two. This pattern shows that evolution learned recurring unemployment cues that the schema-only and refinement-only modes treated as ambiguous. The main Housing error was `Housing` to `Unknown`, which decreased from 10 errors in Schema Mode to three in Evolution Mode and four in Full Mode. Detailed label-level error tables and SDOH confusion matrices are provided in Appendix C.

3.4 Cross-Dataset Mechanism Comparison

The relative value of refinement and evolution depended on task structure. Table 5 compares the main mechanism-level effects across ECG and SDOH. Refinement was most useful for ECG, where the notes contained implicit diagnostic language, synonyms, negation, and ambiguous phrasing. Evolution was most useful for SDOH, where recurring patterns in social history text could be encoded as persistent skills.

The cross-dataset pattern supports a task-dependent interpretation. Query-specific refinement is most valuable when individual notes require local reasoning that may not repeat often. Persistent evolution is most valuable when failures recur across records and can be converted into durable extraction rules. The Full Mode was not automatically superior. On ECG, it regressed relative to both Refinement Mode and Evolution Mode. On SDOH, it matched

Table 5: Cross-dataset comparison of mechanism effects. Values report approximate F1 gain relative to the indicated comparison point.

Mechanism	ECG	SDOH
Schema grounding	+0.1031 over ClinBench, but many records returned no output.	+0.0300 over ClinBench, with no empty outputs because both fields had `Unknown` defaults.
Query-specific refinement	Dominant mechanism, improving F1 from 0.6731 to 0.9524.	No overall F1 gain. Schema and Refinement both achieved 0.8500.
Persistent evolution	Nearly matched refinement, reaching 0.9450 after seven events.	Dominant mechanism, improving F1 from 0.8500 to 0.8955.
Full system	Regressed relative to both adaptive single-mechanism modes.	Matched Evolution Mode but required more tokens and events.

Evolution Mode but did not improve on it. Additional record-level rescue, regression, cost, and skill-evolution analyses are provided in Appendix C and Appendix D.

4 Discussion

This study evaluated MedSkill as a progressive skill enrichment framework for structured clinical information extraction. Across two clinically distinct tasks, the results show that skill-based extraction improves over a static ClinBench GPT-4o baseline, but the most useful skill mechanism depends on the structure of the task. For ECG atrial fibrillation extraction, query-specific refinement was the strongest mechanism, increasing F1 from 0.6731 in Schema Mode to 0.9524 in Refinement Mode. This suggests that ECG reports require note-level reasoning over implicit clinical language, including negation, rhythm synonyms, and ambiguous diagnostic phrases. For SDOH extraction, persistent evolution was the dominant mechanism. Evolution Mode improved overall F1 from 0.8500 to 0.8955 and reduced failures from 30 to 20 records, indicating that recurring social-history patterns can be converted into durable extraction rules.

The results also show that schema grounding is a useful foundation even without adaptation. Schema Mode exceeded the ClinBench baseline on both ECG and SDOH, suggesting that constraining outputs to valid fields and labels reduces the noise associated with unstructured prompting. However, schema grounding behaved differently across tasks. In ECG, Schema Mode achieved high precision but low recall because it often returned an empty output when the note did not contain an explicit `AF` or `NotAF` signal. This refusal behavior reduced F1, but it also reflected a conservative extraction policy that avoided unsupported labels. In SDOH, no mode produced empty outputs because both fields had a valid `Unknown` default. As a result, SDOH errors were wrong-label errors rather than abstentions.

A main finding is that the Full Mode was not consistently superior to the single-mechanism modes. On ECG, Full Mode underperformed both Refinement Mode and Evolution Mode, with F1 decreasing to 0.8895. Record-level analysis showed that Full Mode introduced regressions on records that Refinement and Evolution handled correctly. The skill lifecycle analysis helps explain this pattern. In ECG Full Mode, refinement resolved many early failures, delaying evolution until late in the run. The resulting evolved skill was built from sparse failure evidence and appears to have introduced persistent rules that interfered with otherwise correct per-note reasoning. This result is important because it shows that combining adaptive mechanisms can create interference rather than additive gains when the failure signal is limited or biased.

The ECG experiments also revealed a safety-relevant limitation in the reward signal. The recorded failure count penalized empty outputs and false positives, but false negatives, AF records predicted as `NotAF`, were not consistently represented as failures. This asymmetry affected evolution. Evolution Mode reduced refusals to zero and lowered the recorded failure count, but it also increased false negatives from 1 in Refinement Mode to 6. In clinical AF extraction, this direction of error is concerning because missed AF may be more harmful than false alarms. This finding shows that skill evolution can optimize the signal it receives, even when that signal does not fully match clinical risk. Future versions of MedSkill should use class-sensitive reward functions, explicit confusion-matrix monitoring, and safety gates that penalize clinically dangerous errors during replay.

The SDOH results suggest a different role for evolution. Social-history notes contained repeated patterns, such as relational living descriptions, past-tense employment language, and homelessness cues. Evolution converted these patterns into reusable skill rules and improved both employment and housing extraction. Housing benefited most, with Housing F1 increasing from 0.8727 in Schema Mode to 0.9273 in Evolution Mode. Employment remained harder because `Unemployed` and `Unknown` overlap linguistically when notes contain past employment, disability, retirement, or no explicit work status. The persistent 15 all-mode SDOH failures suggest that some errors reflect label ambiguity or underspecified annotation rules rather than a lack of extraction skill alone.

The cost analysis supports Evolution Mode as the most efficient adaptive setting. For ECG, Evolution Mode achieved nearly the same F1 as Refinement Mode while using fewer tokens per record. For SDOH, Evolution Mode matched Full Mode's F1 while using approximately one-third of the token budget. These results suggest that persistent skill evolution is especially useful when failure patterns recur often enough to justify the cost of skill creation and replay validation. Query-specific refinement is more useful when individual notes require local interpretation that may not generalize across many records. Full Mode may be justified only when both conditions hold: the task has recurring failures and also requires note-specific reasoning.

This study has several limitations. First, the evaluation used two clinical extraction settings, ECG AF diagnosis and SDOH employment and housing extraction. These tasks represent different

language structures, but they do not cover the full range of clinical information extraction problems. Second, all runs used `gpt-4o` with temperature 0, so the results may differ with other models, decoding settings, or agent implementations. Third, the evolution process depends on the reward and validation design. The ECG reward asymmetry shows that an incomplete evaluator can produce harmful skill drift. Fourth, the results were evaluated on fixed retrospective samples rather than prospective deployment data. Future work should test MedSkill across more clinical domains, include non-public and temporally separated datasets, use clinically weighted reward functions, and evaluate whether evolved skills remain valid under dataset shift.

MedSkill therefore supports a broader conclusion about clinical agents. Performance depends not only on the language model, but also on how procedural knowledge is represented, constrained, evaluated, and updated. Schema grounding improves output validity, refinement improves note-level reasoning, and evolution improves recurring-pattern learning. The safest and most efficient configuration depends on the task. Clinical skill systems should therefore be evaluated not only by aggregate F1, but also by failure type, class-specific error direction, cost, and the behavior of the skills that the system learns over time.

References

[1] Hiba Ahsan, Emmie Ohnuki, Avijit Mitra, and Hong You. 2021. MIMIC-SBDH: a dataset for social and behavioral determinants of health. In *Machine Learning for Healthcare Conference*. PMLR, 391–413.

[2] Rahul K. Arora, Jason Wei, Rebecca Soskin Hicks, Preston Bowman, Joaquin Quiñonero Candela, Foivos Tsimpourlas, Michael Sharman, Meghan Shah, Andrea Vallone, Alex Beutel, Johannes Heidecke, and Karan Singhal. 2025. HealthBench: Evaluating Large Language Models Towards Improved Human Health. *arXiv preprint arXiv:2505.08775* (2025). https://arxiv.org/abs/2505.08775

[3] Brian Gow, Tom Pollard, Larry A. Nathanson, Alistair Johnson, Benjamin Moody, Chrystinne Fernandes, Nathaniel Greenbaum, Jonathan W. Waks, Parastou Eslami, Tanner Carbonati, Ashish Chaudhari, Elizabeth Herbst, Dana Moukheiber, Seth Berkowitz, Roger Mark, and Steven Horng. 2023. MIMIC-IV-ECG: Diagnostic Electrocardiogram Matched Subset. *PhysioNet* (Sept. 2023). doi:10.13026/4nqg-sb35 Version 1.0.

[4] Xiangyi Li et al. 2026. SkillsBench: Benchmarking How Well Agent Skills Work Across Diverse Tasks. *arXiv preprint arXiv:2602.12670* (2026). https://arxiv.org/abs/2602.12670

[5] Yuan Liang, Ruobin Zhong, Haoming Xu, Chen Jiang, Yi Zhong, Runnan Fang, Jia-Chen Gu, Shumin Deng, Yunzhi Yao, Mengru Wang, et al. 2026. SkillNet: Create, Evaluate, and Connect AI Skills. *arXiv preprint arXiv:2603.04448* (2026). https://arxiv.org/abs/2603.04448

[6] Songyang Liu, Chaozhuo Li, Chenxu Wang, Jinyu Hou, Zejian Chen, Litian Zhang, Zheng Liu, Qiwei Ye, Yiming Hei, Xi Zhang, and Zhongyuan Wang. 2026. ClawKeeper: Comprehensive Safety Protection for OpenClaw Agents Through Skills, Plugins, and Watchers. *arXiv preprint arXiv:2603.24414* (2026). https://arxiv.org/abs/2603.24414

[7] Timo Schick, Jane Dwivedi-Yu, Roberto Dessì, Roberta Raileanu, Maria Lomeli, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. 2023. Toolformer: Language Models Can Teach Themselves to Use Tools. In *Advances in Neural Information Processing Systems*, Vol. 36. https://arxiv.org/abs/2302.04761

[8] Samuel Schmidgall, Rojin Ziaei, Carl Harris, Eduardo Reis, Jeffrey Jopling, and Michael Moor. 2025. AgentClinic: A Multimodal Agent Benchmark to Evaluate AI in Simulated Clinical Environments. *arXiv preprint arXiv:2405.07960* (2025). https://arxiv.org/abs/2405.07960

[9] Karan Singhal, Shekoofeh Azizi, Tao Tu, S. Sara Mahdavi, Jason Wei, Hyung Won Chung, Nathan Scales, Ajay Tanwani, Heather Cole-Lewis, Stephen Pfohl, et al. 2023. Large Language Models Encode Clinical Knowledge. *Nature* 620, 7972 (2023), 172–180. doi:10.1038/s41586-023-06291-2

[10] Karan Singhal, Tao Tu, Juraj Gottweis, Rory Sayres, Ellery Wulczyn, Mohamed Amin, Le Hou, Kevin Clark, Stephen R. Pfohl, Heather Cole-Lewis, et al. 2025. Toward Expert-Level Medical Question Answering with Large Language Models. *Nature Medicine* 31, 3 (2025), 943–950. doi:10.1038/s41591-024-03423-7

[11] Ismael Villanueva Miranda, Zifan Gu, Donghan Yang, Kuroush Nezafati, Jingwei Huang, Peifeng Ruan, Xiaowei Zhan, Guanghua Xiao, and Yang Xie. 2025. ClinBench: A Standardized Multi-Domain Framework for Evaluating Large Language Models in Clinical Information Extraction. In *Advances in Neural Information Processing Systems*, Vol. 38. https://papers.nips.cc/paper_files/paper/2025/hash/621b4af77315234465ef366ea14b5dc5-Abstract-Datasets_and_Benchmarks_Track.html

[12] Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. 2023. ReAct: Synergizing Reasoning and Acting in Language Models. In *International Conference on Learning Representations*. https://openreview.net/forum?id=WE_vluYUL-X

A Additional Method Details

A.1 Output Constraints

For the ECG task, the output constraints take the following form.

```
## Output Constraints
Output ONLY the following fields (use exact field names):
- diagnosis: one of ['AF', 'NotAF']
Do NOT add any extra fields beyond those listed above.
If a field is not supported by explicit evidence in the note, omit it.
```

For the SDOH task, the output constraints specified two required fields.

```
## Output Constraints
Output ONLY the following fields (use exact field names):
- employment: one of ['Employed', 'Unemployed', 'Unknown']
  (REQUIRED — always output a value; use "Unknown" when
  there is no explicit evidence in the note)
- housing: one of ['Housing', 'Homeless', 'Unknown']
  (REQUIRED — always output a value; use "Unknown" when
  there is no explicit evidence in the note)
Do NOT add any extra fields beyond those listed above.
Fields marked REQUIRED must always appear in the JSON output.
For all other fields, omit if there is no supporting evidence.
```

A.2 Artifact Pipeline

After parsing, MedSkill executes the active skill's artifact pipeline. Table 6 summarizes the deterministic post-processing artifacts and their order of application.

Table 6: Post-extraction artifact pipeline used after JSON parsing.

Artifact	Role
`schema.json`	Canonicalizes field names through aliases, enforces enumerated values, and applies the `additionalProperties` policy.
`transform.py`	Executes `transform(fields, note) -> dict` for deterministic Python-based normalization, usually through regular expressions.
`validator.py`	Executes `validate(fields, note) -> list[str]` to identify consistency problems. Validation errors are logged but do not automatically discard the prediction.

A.3 Evolution Artifacts

Table 7 summarizes the candidate artifact types used by the evolution engine.

Table 7: Candidate artifact types used by the evolution engine.

Artifact type	Typical role in evolution
`skill`	Adds or edits task-specific extraction rules, examples, lexical cues, and gotchas. Preferred when failures involve missing ground-truth fields or recurrent reasoning errors.
`transform`	Adds deterministic normalization rules, often for binary labels or label-string normalization. Implemented as Python regular-expression logic.
`schema`	Enforces field names, enumerated values, aliases, and additional-property behavior. Useful when predictions contain hallucinated fields or invalid labels.
`validator`	Adds consistency checks that detect contradictions between extracted fields and the source note. Validation errors are logged for analysis and future evolution.

Within a triggered cluster, traces are grouped by failing field signature, defined as the sorted tuple of ground-truth fields that failed. Each sub-batch is capped at eight traces to keep the builder prompt within the model context window. First, `proposer.analyze()` receives the failure sub-batch, previous proposal outcomes from `FeedbackHistory`, the task specification, and up to three recent success examples. It produces a proposal specifying whether to create or edit an artifact, the artifact type, the target skill if applicable, a description, a justification, and related prior iterations. The routing guidance favors `transform` artifacts for binary fields and label normalization, `skill` artifacts for missing ground-truth fields, and `schema` artifacts for hallucinated fields.

MedSkill normalizes the proposal target before building the artifact. The universal `extract-general` skill is never edited, because it must remain dataset-agnostic. If the proposer targets `extract-general`, the target is reset and the proposal is converted to a new skill creation. If

the proposer requests a non-skill artifact without a valid non-generic target, the request is promoted to a skill creation. New skill names use an `extract-` prefix followed by one field name, or by two field names joined with `-and-`, and are capped at 64 characters. Generic names such as `skill_N` are overwritten.

If holdout validation is enabled by setting a positive evolution validation size, the updated registry is evaluated on the held-out set. If held-out F1 falls below the previous F1 minus the gate tolerance, the artifact application is reverted and the registry is reloaded from the snapshot. Early stopping freezes evolution after a configured number of consecutive non-improving events.

A.4 LLM Call Structure

Table 8 summarizes the approximate LLM call structure for each operating mode.

Table 8: Approximate LLM call structure by operating mode.

Mode	LLM calls
Schema Mode	One extraction call per record.
Refinement Mode	One refinement call and one extraction call per record.
Evolution Mode	One extraction call per record, plus amortized evolution calls when failure clusters trigger.
Full Mode	One refinement call and one extraction call per record, plus amortized evolution calls when post-refinement failure clusters trigger.

B Dataset and Evaluation Details

B.1 ECG Label Distribution

Table 9 reports the balanced label distribution for the ECG atrial fibrillation task.

Table 9: Label distribution for the ECG atrial fibrillation diagnosis task.

Label	Count	Percentage
AF	100	50%
NotAF	100	50%

B.2 SDOH Label Definitions and Distributions

Table 10 defines the SDOH labels used for employment and housing extraction.

Table 10: Task definitions for the SDOH extraction fields.

Field	Label	Output status	Definition
employment	Employed	Required	The note explicitly indicates current work or student status, including active employment or education.
employment	Unemployed	Required	The note explicitly indicates no current job, retirement, disability, homemaker status, or layoff.
employment	Unknown	Required default	Employment is not mentioned in the note.
housing	Housing	Required	The note indicates stable housing, such as living at home, with family, with a partner, or in supportive housing.
housing	Homeless	Required	The note indicates adverse housing, such as shelter residence, no fixed address, or no stable housing.
housing	Unknown	Required default	Housing is not mentioned in the note and should not be inferred from discharge destination.

Table 11 reports the label distribution for the sampled SDOH evaluation set.

Table 11: Label distribution for the sampled SDOH evaluation set, $n = 110$ with seed=42.

Field	Label	Count	Percentage
employment	Employed	26	24%
employment	Unemployed	28	25%
employment	Unknown	56	51%
housing	Housing	62	56%
housing	Unknown	34	31%
housing	Homeless	14	13%

Table 12 reports the label distribution in the full SDOH source dataset.

Table 12: Label distribution in the full SDOH source dataset, $n = 1{,}405$.

Field	Label	Count	Percentage
employment	Unknown	859	61%
employment	Unemployed	385	27%
employment	Employed	161	11%
housing	Housing	878	63%
housing	Unknown	513	37%
housing	Homeless	14	1%

B.3 Evaluator Error Types and Experimental Configuration

Table 13 lists the evaluator error types.

Table 13: Evaluator error types.

Error type	Definition
`missing`	A ground-truth field is absent from the prediction.
`wrong_value`	A field is present, but its predicted label does not match the ground truth.
`hallucinated`	A field appears in the prediction but not in the ground truth. This is logged but does not reduce the reward.

Table 14 summarizes the experimental configuration.

Table 14: Experimental configuration.

Parameter	Value
Model	`gpt-4o`
Temperature	0 for all LLM calls
Top-k retrieval	3 skills per query
Evolution batch size	5 failures per cluster trigger
Semantic cluster threshold	$\theta_c = 0.72$
Adaptive cluster trigger threshold	$\alpha = 0.88$
Admission minimum improvement	$\Delta_{\min} = 0.02$ mean reward improvement
Pass threshold	$\tau = 1.0$, meaning all ground-truth fields must be correct
ECG sample	$n = 200$, balanced with 100 records per class, `seed=42`
SDOH sample	$n = 110$, two fields extracted jointly, three labels per field, `seed=42`
Record alignment	Identical record sets across modes within each dataset, propagated with `--include-ids-file`
Comparison baseline	ClinBench GPT-4o, using the same model with unstructured prompting, no schema, and no skill library.

C Extended Results Diagnostics

C.1 Confusion Matrices

Figure 5 shows the ECG confusion matrices across modes.

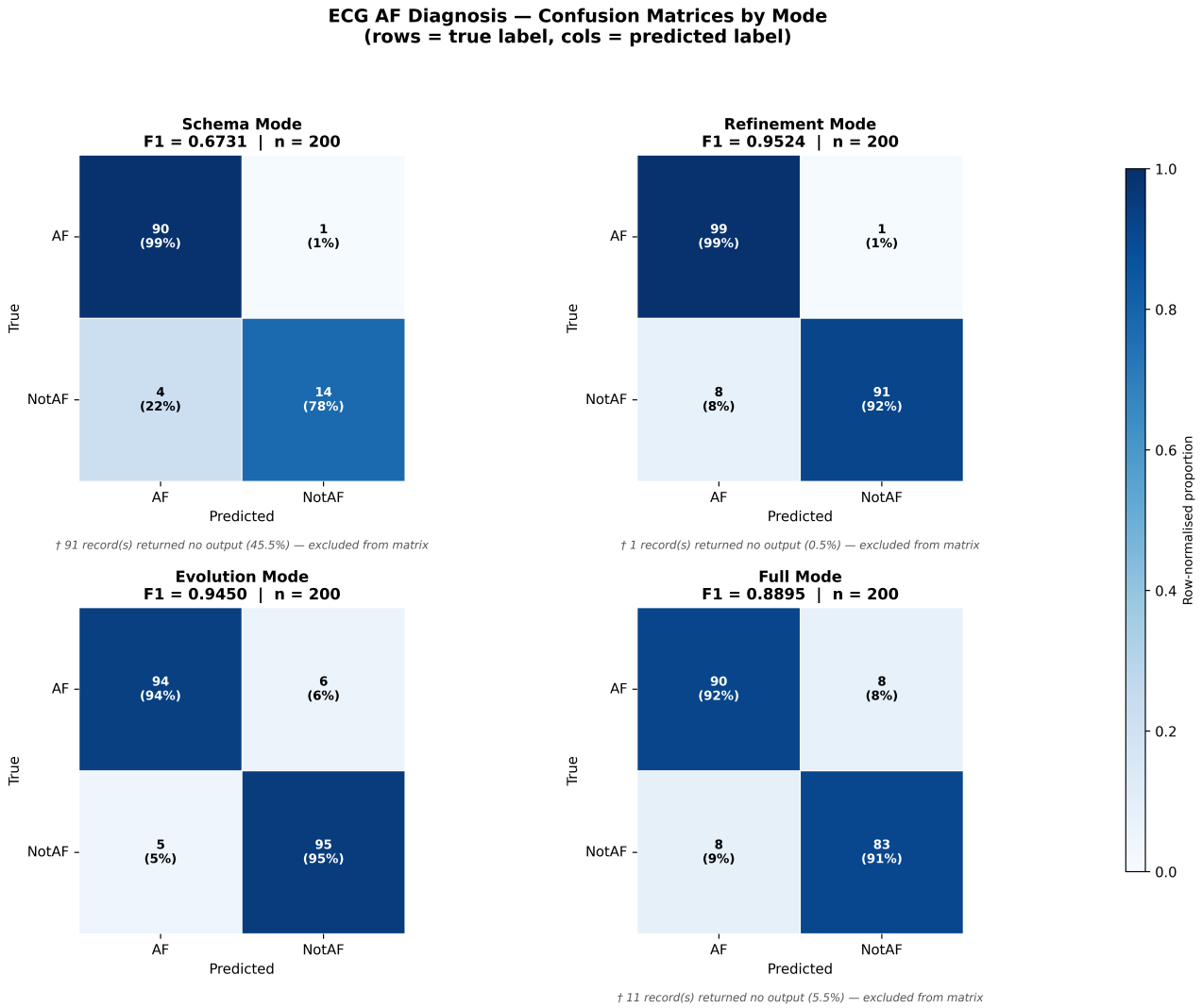

Figure 5: ECG AF diagnosis confusion matrices by mode. Rows show true labels and columns show predicted labels. Matrices are row-normalized, and cell text reports raw counts and row percentages. Empty outputs are excluded from matrix cells and reported as footnotes.

Figure 6 shows the SDOH Employment confusion matrices across modes.

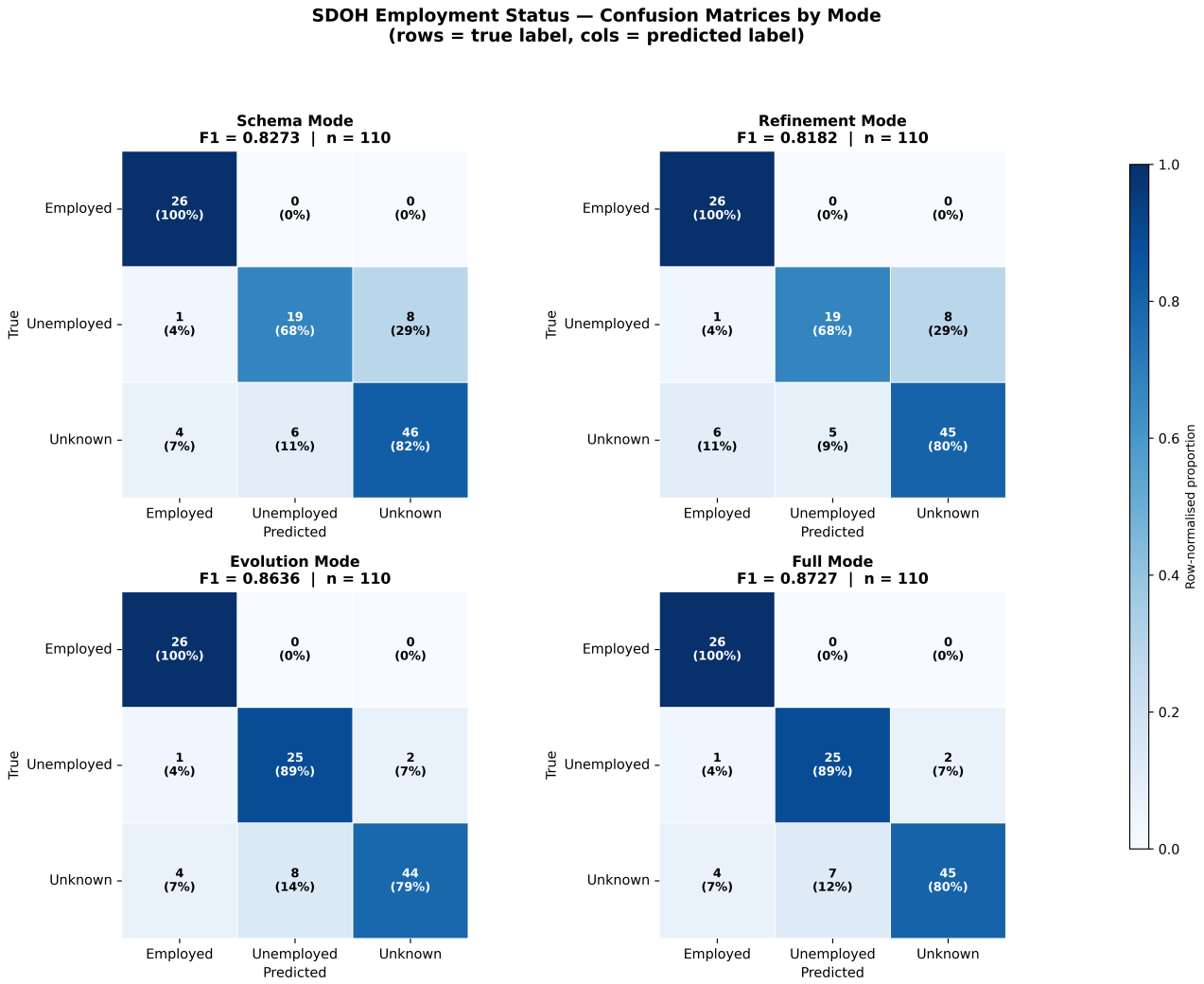

Figure 6: SDOH Employment confusion matrices by mode. Rows show true labels and columns show predicted labels. Matrices are row-normalized, and cell text reports raw counts and row percentages.

Figure 7 shows the SDOH Housing confusion matrices across modes.

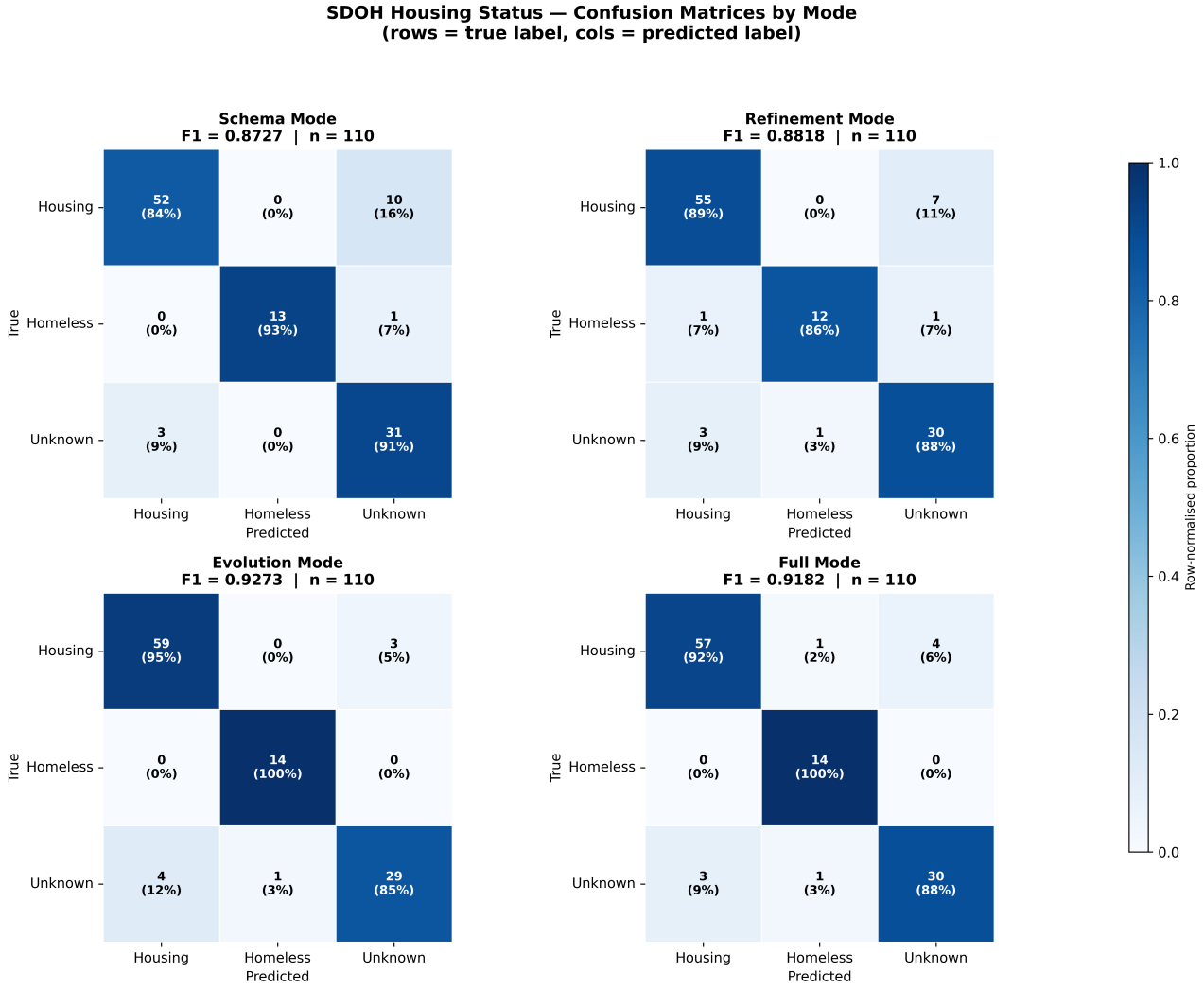

Figure 7: SDOH Housing confusion matrices by mode. Rows show true labels and columns show predicted labels. Matrices are row-normalized, and cell text reports raw counts and row percentages.

Figure 8 aggregates SDOH label-level errors across modes.

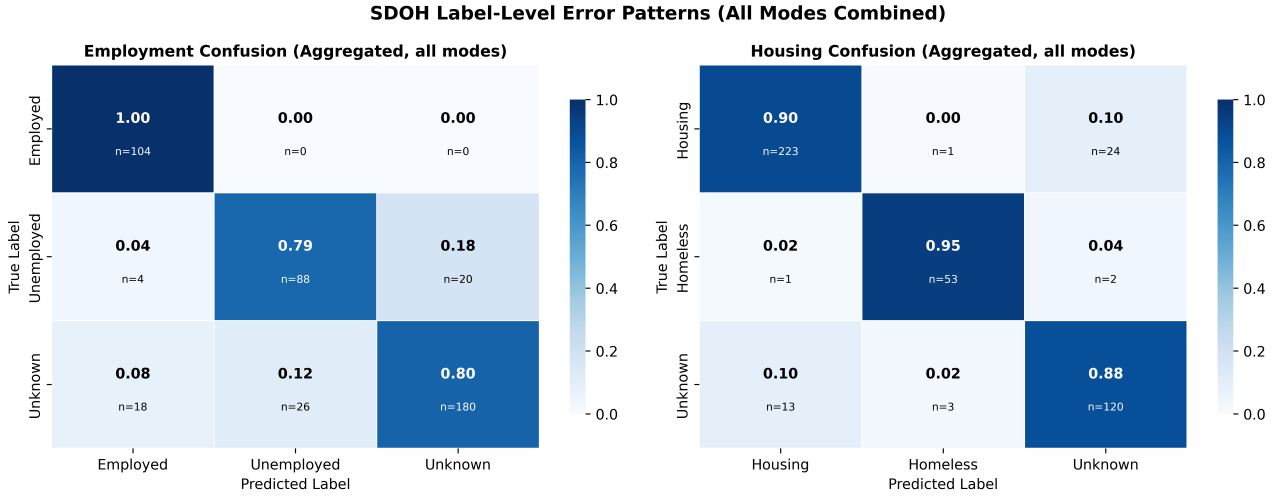

Figure 8: Aggregated SDOH label-level error patterns across all modes. Each heatmap is row-normalized. Cell text reports the normalized rate and the raw count. Employment errors were more dispersed, while Housing errors were concentrated mainly in `Housing` to `Unknown`.

C.2 SDOH Label-Level Errors

Table 15 reports SDOH label-level error patterns by mode.

Table 15: SDOH label-level error patterns by mode. Counts are derived from direct label comparisons for the `employment` and `housing` fields.

Mode	Record failures	Employment fails	Housing fails	Unemp. to Unknown	Unknown to Emp.	Unknown to Unemp.	Housing to Unknown	Unknown to Housing
Schema	30	19	14	8	4	6	10	3
Refinement	30	20	13	8	6	5	7	3
Evolution	20	15	8	2	4	8	3	4
Full	20	14	9	2	4	7	4	3

C.3 Cross-Mode Record-Level Rescue and Regression

Table 16 reports ECG record-level rescue and regression patterns.

Table 16: ECG cross-mode record-level rescue and regression analysis.

Comparison	Count	Interpretation
Records failing all four modes	2/200	These records were ambiguous across all conditions.
Schema failures rescued by Refinement	86/95	Refinement resolved 91% of Schema Mode failures.
Schema failures rescued by Evolution	90/95	Evolution resolved 95% of Schema Mode failures.
Schema passes to Refinement failures	0	Refinement did not degrade records that Schema Mode handled correctly.
Refinement passes to Full failures	15	Full Mode introduced regressions relative to Refinement.
Evolution passes to Full failures	17	Full Mode introduced regressions relative to Evolution.

Table 17 reports SDOH record-level rescue and regression patterns.

Table 17: SDOH cross-mode record-level rescue and regression analysis.

Comparison	Count	Interpretation
Records failing all four modes	15/110	These records formed a persistent hard-case set.
Schema failures rescued by Refinement	3/30	Refinement had limited rescue effect on SDOH.
Schema failures rescued by Evolution	13/30	Evolution was the main rescue mechanism.
Schema passes to Refinement failures	3	Refinement introduced a small number of regressions.
Evolution passes to Full failures	3	Full Mode introduced minimal additional interference on SDOH.

Figure 9 shows ECG failure overlap across modes.

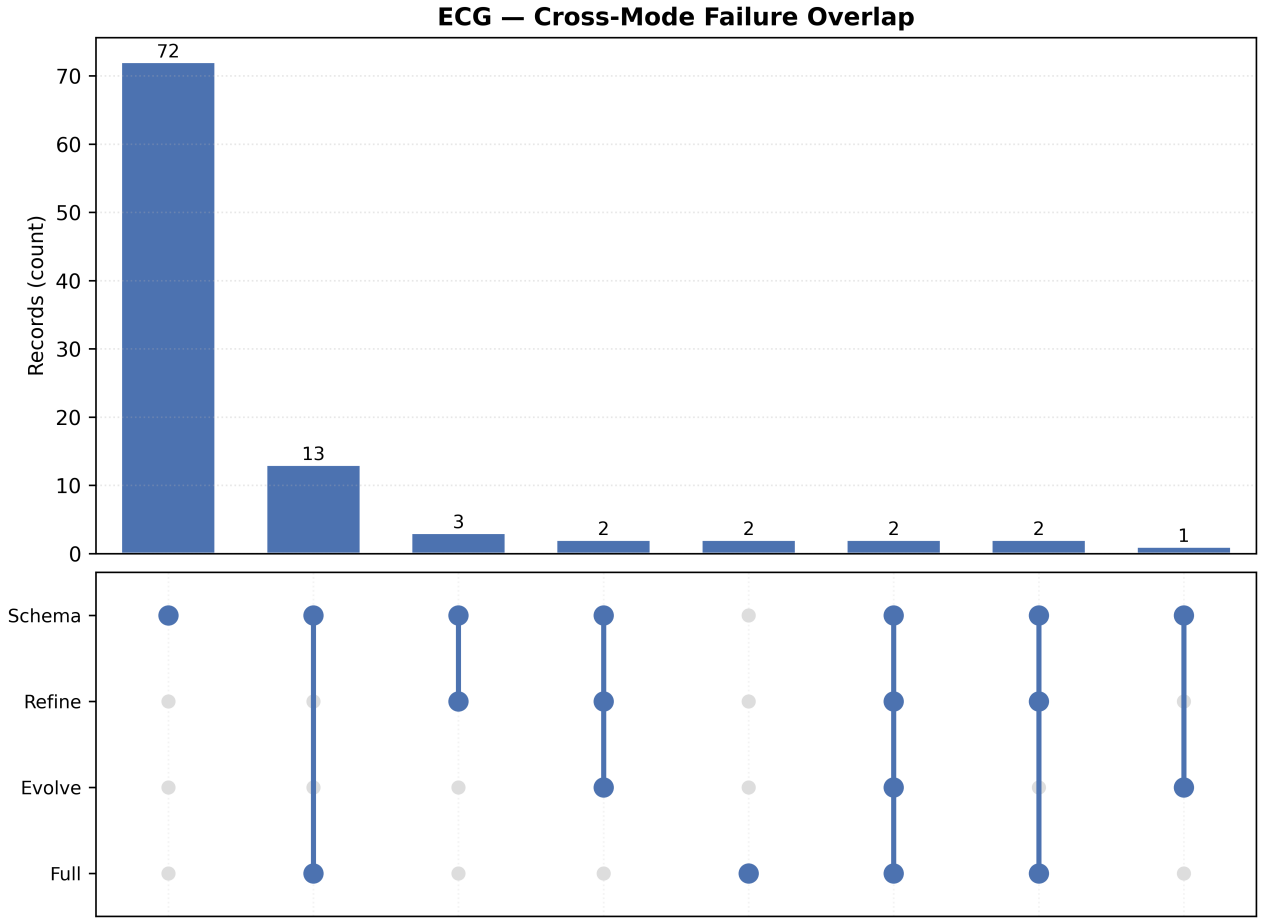

Figure 9: ECG cross-mode failure overlap. The top bars show the number of records in each exact failure intersection, and the lower dot matrix identifies the modes included in each intersection.

Figure 10 shows SDOH failure overlap across modes.

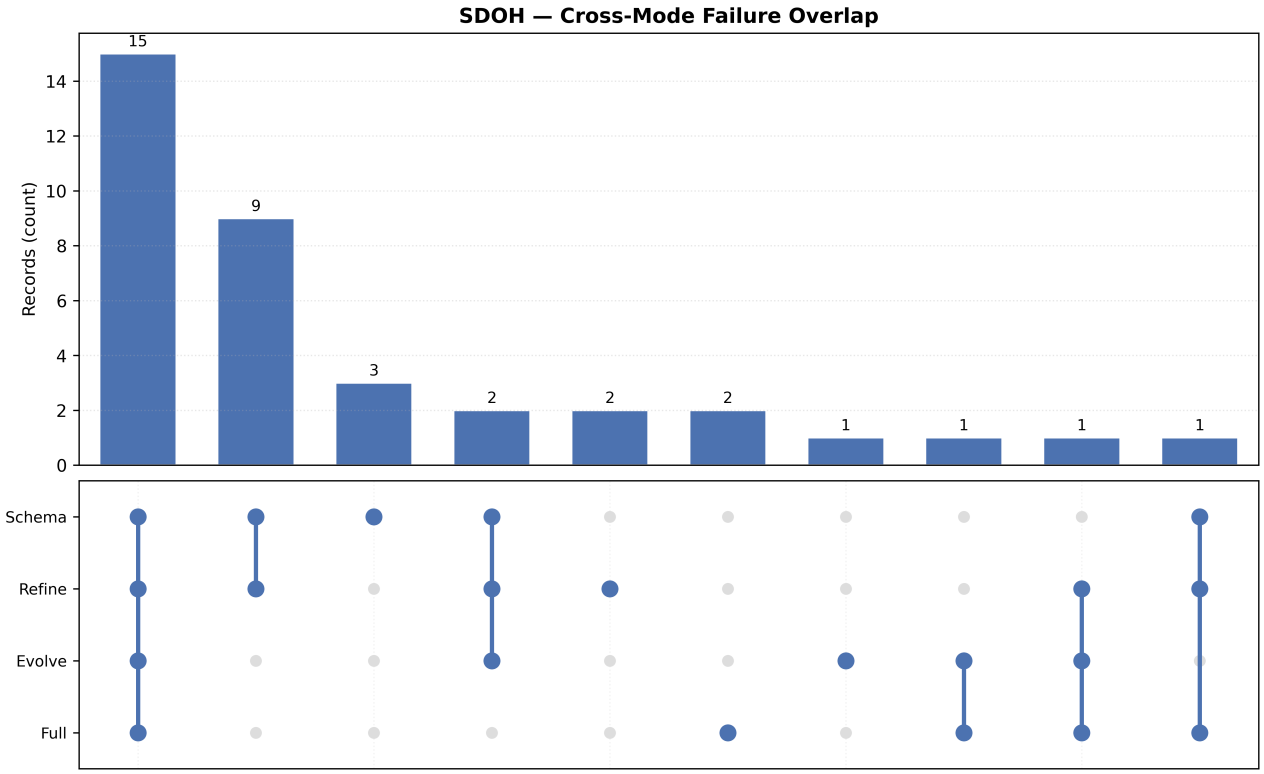

Figure 10: SDOH cross-mode failure overlap. The diagram shows that SDOH contained a larger group of records that failed under all four operating modes, consistent with persistent label ambiguity.

D Cost, Skill Evolution, and Run Registry

D.1 Computational Cost

Tables 18 and 19 report computational cost by mode.

Table 18: ECG computational cost by mode.

Mode	Total tokens	Tokens/record	LLM calls	Duration
Schema	154,413	772	200	182 s
Refinement	605,851	3,029	400	1,943 s
Evolution	449,349	2,246	432	327 s
Full	1,449,854	7,249	826	4,270 s

Table 19: SDOH computational cost by mode.

Mode	Total tokens	Tokens/record	LLM calls	Duration
Schema	133,300	1,211	110	82 s
Refinement	440,903	4,008	220	1,035 s
Evolution	517,863	4,707	272	570 s
Full	1,563,502	14,213	534	2,981 s

D.2 Skill Lifecycle and Learned Skill Content

Figure 11 shows skill lifecycle decisions for evolution-enabled runs.

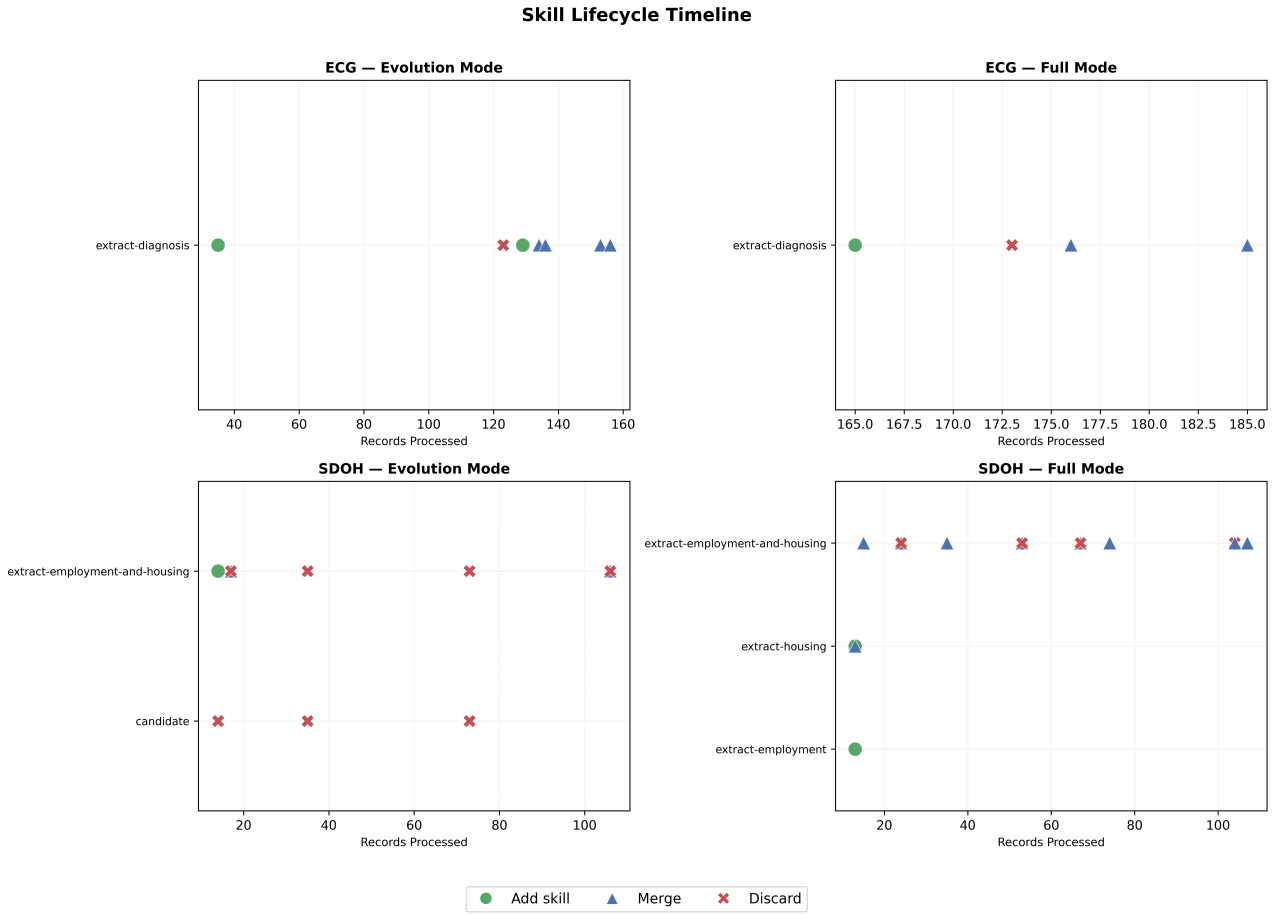

Figure 11: Skill lifecycle timeline for evolution-enabled runs. Each marker shows an evolution decision at the corresponding number of processed records. Circles indicate added skills, triangles indicate merges, and crosses indicate discarded candidates.

Table 20 summarizes learned skill content from evolution-enabled runs.

Table 20: Summary of learned skill content from evolution-enabled runs.

Dataset and mode	Learned patterns	Observed effect
ECG Evolution Mode	Positive triggers included atrial fibrillation, atrial flutter, flutter waves, and atrial-context conduction language. Anti-patterns included isolated ST-T abnormalities and ventricular pacing without atrial arrhythmia evidence.	Reduced empty outputs to zero and reached 0.9450 F1, but increased AF to `NotAF` false negatives because of a conservative default rule.
SDOH Evolution Mode	Learned joint rules for employment and housing, including past-tense employment as `Unemployed`, facility residence as `Housing`, relational living as `Housing`, and discharge-context exclusion for generic "home" mentions.	Reduced record failures from 30 to 20 and improved overall F1 from 0.8500 to 0.8955.
SDOH Full Mode	Added additional employment and housing patterns, including homemaker language, daycare participation as not employment, implicit household evidence, and more precise formerly or used-to-work rules.	Matched Evolution Mode in overall F1 but required more evolution events and substantially higher token cost.

D.3 Run Registry

Tables 21 and 22 list the final run identifiers.

Table 21: Final ECG run registry. All ECG runs used `gpt-4o`, temperature 0, and a balanced $n = 200$ sample.

Condition	Run ID	F1	P	R	Failures
Schema Mode	`20260519_094331`	0.6731	0.9541	0.5200	95
Refinement Mode	`20260519_094810`	0.9524	0.9548	0.9500	9
Evolution Mode	`20260519_115012`	0.9450	0.9450	0.9450	5
Full Mode	`20260519_102446`	0.8895	0.9153	0.8650	19

Table 22: Final SDOH run registry. All SDOH runs used `gpt-4o`, temperature 0, and an $n = 110$ sample.

Condition	Run ID	F1	Employment F1	Housing F1	Failures
Schema Mode	`20260519_122631`	0.8500	0.8273	0.8727	30
Refinement Mode	`20260519_123425`	0.8500	0.8182	0.8818	30
Evolution Mode	`20260519_130054`	0.8955	0.8636	0.9273	20
Full Mode	`20260519_132734`	0.8955	0.8727	0.9182	20