Agent–Augmented Q–Learning for Healthcare Reporting

---

1. Overview

We are developing a **hybrid reinforcement–learning (RL) system** that:

1. **Compares** traditional **pure Q–learning** against
2. **Agent–augmented Q–learning** (using CrewAI agents to shape context, actions, and rewards),

and then demonstrates the **agentic RL** approach in a **healthcare reporting** application.
Our pipeline includes:

- **Synthetic data generation** to create realistic patient trajectories.
- **Offline RL modules** for pure Q–learning training and evaluation.
- **CrewAI–based RL Flow** for agentic training.
- A **reporting flow** that generates patient risk reports (prediction + SHAP explanations).
- **Integration** of RL into the reporting flow: the RL agent dynamically tunes report configuration based on patient outcomes.

---

2. Synthetic Data Generation

2.1 Anchors & Trajectories

- We extract **anchor pairs** $(p_{\text{start}}, p_{\text{end}})$ by sampling "Present" and "Absent" entries from the original Heart Disease CSV.
- Each anchor pair seeds one **base trajectory**:

$$\{p_0, p_1, \ldots, p_{T-1}\} \quad \text{drifting smoothly from } p_0 \to p_{T-1}.$$

- **Rationale**: Anchors model realistic inflection points (onset/remission) in patient risk.

2.2 Drift Model

We model the drifted risk probability at time t by

$$\tilde{p}_t = \bigl(1 - \tfrac{t}{T-1}\bigr)\, p_{\text{start}} + \tfrac{t}{T-1}\, p_{\text{end}} + \varepsilon_t, \quad \varepsilon_t \sim \mathcal{N}(0, \sigma^2).$$

We then enforce the valid probability range by clipping:

$$p_t = \min\{\max\{\tilde{p}_t,\, 0\},\, 1\}.$$

- **Gaussian noise** ε_t with $\sigma \approx 0.02$ introduces stochasticity.
- No action noise at this stage—pure drift only.

2.3 Action-Dependent Noise

At decision time, the agent's discrete action a_t induces an additive perturbation. Define

$$\delta_t \sim \text{Uniform}\big(\delta_{\min},\, \delta_{\max}\big), \quad \alpha_t = \begin{cases} -1, & \text{if } a_t \text{ starts with "DECREASE"}, \\ +1, & \text{if } a_t \text{ starts with "INCREASE"}, \\ 0, & \text{otherwise (NOP)}, \end{cases}$$

and let

$$\hat{p}_{t+1} = \tilde{p}_{t+1} + \alpha_t\, \delta_t.$$

Finally we clip again into $[0, 1]$:

$$p_{t+1} = \min\{\max\{\hat{p}_{t+1},\, 0\},\, 1\}.$$

- This lets each discrete action (threshold shifts, SHAP-verbosity changes) have a measurable effect on the patient's risk.
- **Note:** action-dependent noise is used here only for evaluation; it would be removed in a real medical deployment.

Mathematicaly speaking:

$$\delta \sim \text{Uniform}(\delta_{\min},\, \delta_{\max}) \quad \text{(e.g. 0.01–0.05)},$$

$$p_{t+1} = \begin{cases} p_{\text{base}} - \delta, & \text{if } a_t \text{ starts with "DECREASE"}, \\ p_{\text{base}} + \delta, & \text{if } a_t \text{ starts with "INCREASE"}, \\ p_{\text{base}}, & \text{otherwise (NOP)}, \end{cases} \qquad p_{t+1} = \text{clip}(p_{t+1},\, 0,\, 1).$$

- Allows the agent's discrete actions (threshold shifts, SHAP verbosity changes) to have a **measurable effect** on patient risk.

---

3. Reinforcement–Learning Framework

3.1 Markov Decision Process

- **States** s_t: discretized risk $p_t \in [0, 1]$ into 10 bins (0 . . . 9).
- **Actions** a_t (7 total):
  - `INCREASE_MODERATE_THRESHOLD` / `DECREASE_MODERATE_THRESHOLD`
  - `INCREASE_HIGH_THRESHOLD` / `DECREASE_HIGH_THRESHOLD`
  - `INCREASE_TOP_K` / `DECREASE_TOP_K`
  - `NOP`
- **Rewards** r_t:

  In our MDP, rather than using a flat $\pm 1/0$ reward, we define the reward as the **actual change in risk probability** (scaled to integer rewards). Concretely:

  $$r_t = \big\lfloor\, (p_t - p_{t+1}) \times M \big\rfloor$$

  - If the patient's predicted risk *drops* by 0.20, then

    $$r_t = \lfloor 0.20 \times M \rfloor = +2.$$

  - If the risk *rises* by 0.05, then

$$r_t = \lfloor -0.05 \times M \rfloor = -0 \quad (\text{or } -1 \text{ if you choose to round away from zero}).$$

  - No change $\rightarrow r_t = 0$.

Here, M is a **scaling factor** matching the number of discrete bins you choose. In our original design $M = 10$, but one might find it wiser to increase bin granularity—e.g. $M = 20$ or $M = 50$—so that small changes in probability produce meaningful nonzero rewards and the agent receives **richer feedback** at each step.

3.2 Pure Q–Learning

- **Offline training** via `train_q_table`:

$$Q[s, a] \leftarrow Q[s, a] + \alpha\big[r + \gamma \max_{a'} Q[s', a'] - Q[s, a]\big]$$

- **Exploration policies**: ε–greedy, softmax, UCB, Thompson sampling, etc.
- **Evaluation**: freeze Q–table, run greedy policy on held–out drift trajectories.

3.3 Agent–Augmented Q–Learning

- **RLEngine** implements:
  - `encode_prev_state`, `encode_curr_state`
  - `compute_env_reward` (pure reward)
  - `generate_context` (via a **ContextProvidingAgent**)
  - `shape_action` (via a **PolicyAgent**)
  - `shape_reward` (via a **RewardShapingAgent**)
  - `save_state` (updates Q–table)
- **RLFlow** (CrewAI Flow) orchestrates the MDP step:
  1. Encode previous state
  2. Encode current state
  3. Compute raw reward
  4. Generate context (agentic summary)
  5. Shape action (agentic policy)
  6. Shape reward (agentic reward)
  7. Update Q–table
- **Offline evaluation**: after training on synthetic episodes, we extract the learned Q–table and evaluate it exactly like pure Q–learning.

---

4. Modules & Directory Layout

Project Structure

For reference when looking at the repository.

```
Health-Coach/
 data/           # raw and synthetic datasets
 knowledge/      # domain documents (e.g. risk guidelines)
 models/         # trained ML models (e.g. logistic regression)
 out/            # experiment outputs, reports, logs
 resources/      # ancillary files (e.g. diagrams)
 pyproject.toml  # project metadata
 README.md       # high-level overview
 src/
     health_coach/
         agents.py        # Agent classes for context, policy, reward shaping
         crew.py          # High-level orchestration of CrewAI agents
         flows.py         # Definition of the RLFlow (step-by-step MDP flow)
         main.py          # Entry point / experiment runner
         rl.py            # RLEngine and QLearningEngine implementations
         rl_data_gen/     # Synthetic data & drift-model generation
         rl_train/        # Offline and online Q-learning training scripts
         compare/         # Comparison harness for pure vs. agentic RL
         tasks.py         # Scheduled-task definitions (automations)
         tools/           # CrewAI @tool definitions (action, context, reward)
         tests/           # Unit and integration tests
```

---

5. Healthcare Reporting Integration

Currently the reporting pipeline operates on a simple logistic regression model sourced from github. Once the final prototype is built, it may be wise to swap in a more mature model, perhaps the one Sham's has developed, or another. The RL and reporting flow is more or less independent of any specific prediction model, only the prediction, and explanatory tool's used by the agents would need to be refactored.

5.1 Reporting Pipeline

1. **Input**: 13-element patient feature vector.
2. **Prediction**: scikit-learn logistic regression → p_{risk}.
3. **Explanation**: SHAP value extraction → feature-level contributions.
4. **Report generation**: HTML template filled with risk, top-k SHAP, narrative.

5.2 RL Flow Coupling

- **Config file** (`config.yml`) controls:
  - `thresholds.moderate`, `thresholds.high`
  - `explanation.top_k`
- The **RLFlow** runs *concurrently* with the reporting flow:
  - **State** = discretized p_{risk} at each patient visit.
  - **Action** = config adjustment (e.g. raise "high" threshold if patients keep getting flagged).
  - **Reward** = did the patient's predicted risk improve on next visit?

- The **report** and **RL** loops communicate via:
  - Updated YAML written by the RL agent
  - Patient history CSV appended with (State, Action, Reward, Next_State)

5.3 RL + Reporting Agents

- **DataLoaderAgent** (`data_input_agent`)
  Responsible for reliably loading structured data (CSV, JSON, database) into the pipeline, validating schema and retrying on I/O errors.
- **DataExportAgent** (`data_export_agent`)
  Handles atomic, error-tolerant persistence of structured outputs (reports, logs, Q-tables) to disk or external stores.
- **PolicyAgent** (`policy_agent`)
  Acts as the RL policy oracle, selecting the next discrete action index based on the current encoded state.
- **ContextProvidingAgent** (`context_providing_agent`)
  Gathers and summarizes per-step contextual information (visit counts, reward trends, Q-table stats) into a concise JSON for downstream shaping.
- **RewardShapingAgent** (`reward_shaping_agent`)
  Inspects each transition's raw reward and uses context tools (visitation counts, momentum, progress) to produce an augmented reward that accelerates learning.
- **PredictionAgent** (`prediction_agent`)
  Loads a trained model and, given patient features, computes the heart-disease risk probability.
- **ExplanationAgent** (`explanation_agent`)
  Uses SHAP to generate a detailed HTML explanation of the model's prediction, highlighting top feature contributions.
- **ReportingAgent** (`reporting_agent`)
  Fills an HTML medical-report template with features, risk score, and explanation, then saves the report file to disk.
- **NOTE**: When the reporting system and RL system are merged into an application, some of these agents may be dropped. For example the *DataLoaderAgent*, *DataExportAgent*, and *PredictionAgent* do nothing more than call a function—might as well call those functions directly.

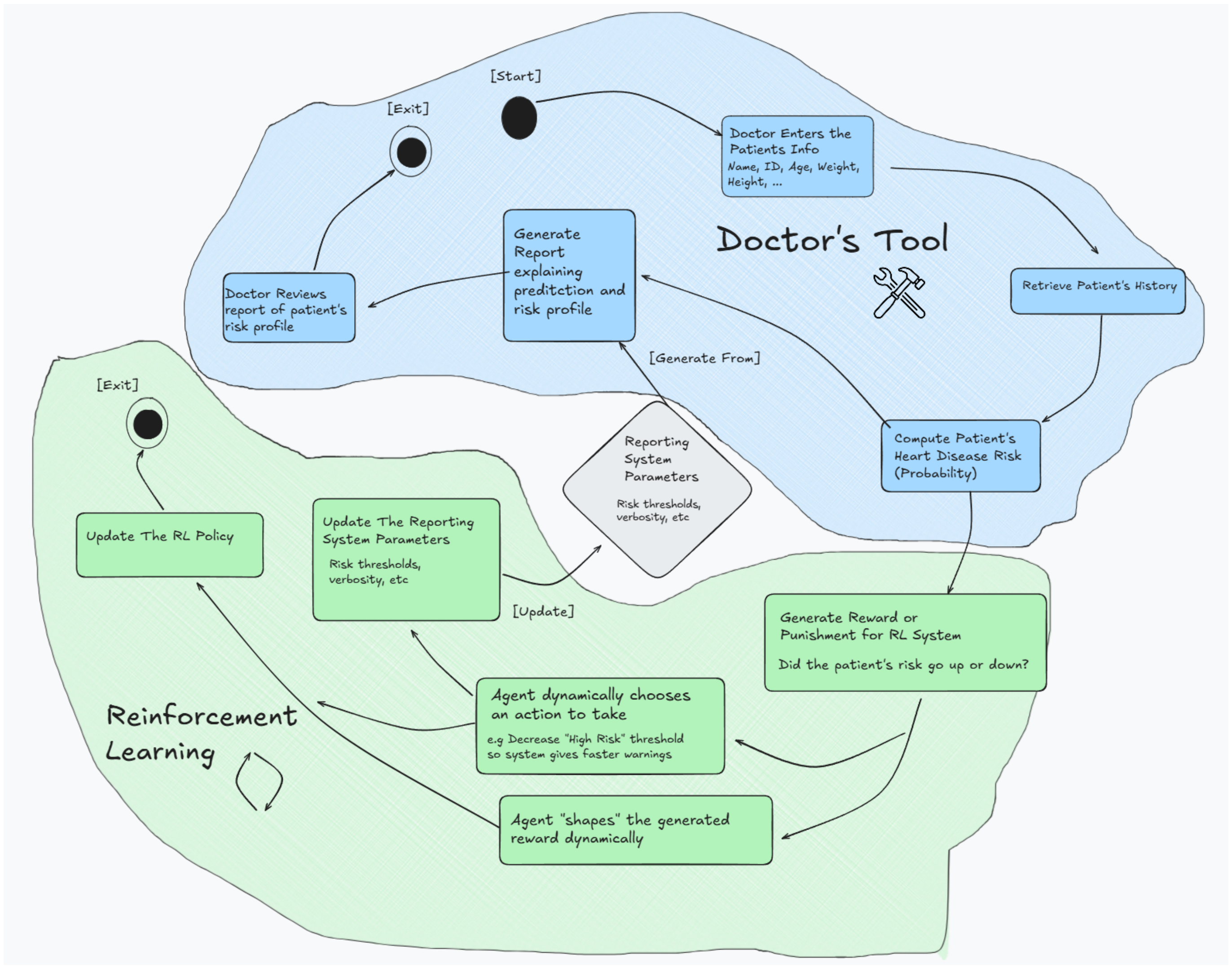
[Start]
[Exit]
Doctor Enters the Patients Info
Name, ID, Age, Weight, Height, ...
Doctor's Tool
Retrieve Patient's History
Generate Report explaining preditction and risk profile
Doctor Reviews report of patient's risk profile
[Generate From]
Reporting System Parameters
Risk thresholds, verbosity, etc
Compute Patient's Heart Disease Risk (Probability)
[Exit]
Update The RL Policy
Update The Reporting System Parameters
Risk thresholds, verbosity, etc
[Update]
Generate Reward or Punishment for RL System
Did the patient's risk go up or down?
Reinforcement Learning
Agent dynamically chooses an action to take
e.g Decrease "High Risk" threshold so system gives faster warnings
Agent "shapes" the generated reward dynamically

Figure 1: Flow chart representing the integration of the *smart reporting system*, that is the RL Flow + the Reporting FLow. The RL flow optimizes the reporting flow behind the scenes. The reporting flow generates it's reports based on a dynamic configuration file.

6. Agent–Augmented Q–Learning: Rationale & Mechanisms

6.1 Why Augment Q–Learning with Agents?

1. **Static vs. Dynamic Adaptation**
2. **Leverage Domain & Meta-Knowledge**
3. **Finer-Grained Control**
4. **Modular & Extensible**

6.2 Exploration Tools & Their Roles

Tool	**Purpose**	**When to Use**
ε-Greedy	With probability ε select a random action; otherwise select $\arg\max_a Q(s,a)$.	Basic baseline; good for steady exploration/exploitation trade-off.
Softmax	Sample actions with probability proportional to $\exp\big(Q(s,a)/T\big)$.	Smooth exploration; temperature T controls randomness.
UCB	$\arg\max_a \Big[Q(s,a) + c\sqrt{\dfrac{\ln N(s)}{N(s,a)+1}}\Big].$	Balance exploration based on visit counts; good for sparse visits.
Count-Bonus	$\arg\max_a \Big[Q(s,a) + \dfrac{\beta}{\sqrt{N(s,a)+1}}\Big].$	Rewards under-visited (s,a) pairs; accelerates coverage.
Thompson Sampling	Draw $Q'(s,a) \sim \mathcal{N}\big(Q(s,a),\sigma\big)$, then pick $\arg\max_a Q'(s,a)$.	Probabilistic exploration; adapts to uncertainty.
MaxEnt	Equivalent to Softmax with an entropy term at temperature α.	Emphasizes maximum entropy; robust against premature convergence.

6.3 Reward-Shaping Tools & Their Benefits

6.4 Context-Providing Tools

Agents shaping context can query the current episode via:

- `get_episode_length`
- `get_moving_average`
- `get_visit_count`, `get_transition_count`
- `get_q_table`

By ingesting this context, a ContextAgent can summarize "State 5 has only been visited twice this episode" or "Mean reward over last 5 steps is –0.2," and feed that into later shaping decisions.

Tool	Transformation	Benefit
Visitation Count Bonus (`shape_vcb`)	$r \mapsto r \; + \; \dfrac{\eta}{\sqrt{N(s)+1}}$	Encourages visiting rare states; combats state neglect.
Rare Transition Bonus (`shape_rtb`)	$r \mapsto r \; + \; \dfrac{\nu}{N(s, a \to s')+1}$	Rewards novel transitions $(s, a \to s')$; uncovers hidden dynamics.
Trend Shaper (`shape_trend`)	$r \mapsto r \; + \; \tau\big(\mathrm{MA}_t - \mathrm{MA}_{t-1}\big)$	Leverages reward momentum; amplifies consistent improvement.
Progress Shaper (`shape_progress`)	$r \mapsto r \times \dfrac{\text{timestep}}{\text{episode length}}$	Increases learning pressure as the episode advances.
Amplify (`shape_amplify`)	$r \mapsto r \times \text{factor}$	Magnifies or tempers the reward signal.

6.5 Continuous Augmentation in Action

Imagine an RL run where:

1. **Early Training**: the ContextAgent detects high Q-uncertainty (via synthetic posterior stdevs). The PolicyAgent chooses Thompson Sampling to aggressively explore promising arms.
2. **Mid-Training**: after 100 episodes, the TrendingAgent observes diminishing reward improvements. It switches to ε-greedy with $\varepsilon = 0.01$ for stable exploitation.
3. **Late Training**: VisitCountBonus is turned on to ensure corner-case states (e.g. s=9) are still sampled at least once before termination.

None of this would be possible with static schedules; agents inject semantic, per-step adjustments based on both global and local context.

6.6 Why Q-Learning?

- **Simplicity**: one tabular update rule, easy to inspect and debug.
- **Transparency**: Q-values clearly map state to action values, ideal for healthcare explainability.
- **Extensibility**: we layer our agentic tools on top without changing the core algorithm, ensuring the baseline remains solid and interpretable.

---

7. Evaluation & Next Steps

The work that needs to be completed:

1. **Run batch comparison** (`main_compare.py`):
    - Pure Q-learning vs. Agent-augmented Q-learning

    - Multiple random seeds, train/val split on synthetic episodes
    - Metrics: mean cumulative reward, policy stability, action distributions

2. **Merge flows**: combine reporting and RL into one CrewAI process with a simple UI and persistent storage (csv, small db, crewai's `@persist` decorator).

3. **Demo**: interactive dashboard showcasing "agentic RL in healthcare"—clinician enters features, sees risk report, config adjusts over time to optimize patient outcomes.

---

8. Rationale & Expected Impact

- **Synthetic trajectories** grounded in real anchor pairs ensure RL trains on plausible clinical progressions.

- **Agentic augmentation** taps domain knowledge (via context & reward-shaping agents) to accelerate learning and improve policy quality.

- **Modular design** (tool–task–agent–flow) makes the system extensible to other healthcare settings (e.g. diabetes management, readmission risk).

- **Clinical demo** will illustrate how an "intelligent" report system can adapt thresholds and explanations over time to better guide clinician decisions.