Learning From One Bit of Feedback in a Modular Hierarchical rApp

Joss Armstrong
Ericsson
Athlone, Ireland
joss.armstrong@ericsson.com

***Abstract*—An energy-saving rApp decides every 15 minutes whether a capacity cell can be switched off. We build it as a modular hierarchy whose components learn to choose switching strategies, load thresholds and context sources. Only the top-level component receives the outcome of the rApp's final switching decision. According to that outcome, it raises or lowers the weight of the component it selected. Each lower component learns from whether the weight assigned to it by the component above increased or decreased, then updates the weights for its own choices. This carries one bit of feedback down the selected path while each component keeps its context and internal choices to itself. A simulated weather forecast service illustrates how a component with its own internal hierarchy uses the same feedback to tune its choices to the operator's switching priorities. The forecast service doesn't receive the rApp's decision record or its scoring rule. We evaluate the hierarchy by replaying 83 days of public counters from 277 capacity cells, with simulated conditions and simulated forecast products. The replays demonstrate adaptation away from a failing context source and tuning of forecast responses to an operator prioritising avoidance of wrong switch-offs. Across different component contexts and learning rules, the hierarchy reproduces the decisions and learned weights obtained with explicit outcome feedback, even when feedback arrives only after each 15-minute interval closes.**

***Index Terms*—O-RAN, rApp, energy saving, learning automata, hierarchical learning, context sources**

I. Introduction

Energy-saving rApps decide when capacity cells can be switched off. The BeGREEN intelligence plane supports the decision with a learned traffic predictor [1]. External context matters too. CAMINO mediates network intents with weather, road-traffic and rollback data, and can reject a power reduction because of adverse weather [2]. Its rules for using that context are fixed in advance. The sources behind them are not. A feed can lose accuracy or stop responding, and a rule that waits for it then stops switching cells off altogether. The rApp should learn how far to trust each source as its usefulness and relevance change, and keep deciding when one fails.

We build the rApp as a hierarchy of selectors. A selector is a component that holds a weight for each of its options and draws one with probability equal to its weight. A decision can pass through any number of selectors. In our base example, the longest path has three, choosing a switching strategy, a load threshold and a context source. The completed switching decision is scored, and only the top-level selector, the root, receives that outcome explicitly. Each lower selector on the path learns from one bit, whether the weight its parent assigned to it rose or fell. We call this the weight-change bit. The selector then updates the weights for its own options. Feedback follows the selected path down the hierarchy, while each selector keeps its context and internal alternatives local.

A component in this hierarchy can have its own internal selectors. We illustrate this with a simulated weather forecast service that chooses between a radar nowcast and a regional forecast, each with several alert thresholds. These choices determine how readily it advises the rApp to hold capacity on. A missed weather event can cause a wrong switch-off, while a false alarm can prevent a worthwhile one. The service needs to tune its responses to the consequences for the rApp. It learns its product and alert threshold from the same weight-change bit as the selectors inside the rApp, without receiving the operator's scoring rule or switching-decision record.

The learning method is not new. A recent result on hierarchies of learning automata [3] concerns selectors that each raise the weight of the option they chose after a good outcome and lower it after a bad one. It shows that the direction of that change carries the outcome to the chosen option, and that the same holds at every level below, whatever update rule or context map each selector uses. Section IV states the two update rules we use and how the direction is read. This paper applies that result to energy saving and evaluates the application by replaying 83 days of public counters from 277 capacity cells [4], [5]. It contributes an application architecture and its evaluation:

- A modular rApp that learns strategy, threshold and source choices through the same one-bit interface at each level. Selectors retain their own contexts and alternatives. Its source selectors adapt as a source degrades and fails, using their ordinary updates without a failure detector (Sections III and VI-B).
- A component with its own internal hierarchy and learning rule, illustrated by a simulated weather forecast service choosing products and alert thresholds. The service tunes its responses to the rApp's switching priorities using the weight-change bit, while the scoring rule and internal choices remain local to their respective components (Sections V and VI-A).
- An implementation that retains each selected path until its outcome becomes available after the 15-minute interval closes. Replays with the forecast service, each paired with a copy given the outcome at every selector, reproduce

the decisions and learned weights of explicit outcome feedback across different contexts and learning rules (Section VI-C). A reproduction package accompanies the paper [6].

In this example, the modular rApp calls a separately maintained forecast service. Each component keeps its own inputs, contexts, alternatives and learning rule. The rApp uses load, load-shape and time contexts. A forecast service could use the direction of incoming weather as its own context, learning to favour radar where coverage is good and another product when weather approaches over the sea. The rApp evaluates the completed switching decision after the 15-minute interval closes, so each selected component must retain its local choice until feedback arrives. These are the information and timing constraints modelled in the replay.

The design question is how to support learning through a common interface under these constraints. Each caller returns the direction of its weight change for the component it selected. That component uses the bit to update its own choices and passes the direction of that update to the next selected component. The forecast service therefore learns through the same interface as the rApp's internal selectors, without exchanging contexts, scoring rules or internal choices. The evaluation checks that this composition preserves explicit-outcome learning and supports useful source and forecast adaptation. All of the rApp's selectors belong to one rApp, whose switching logic stays within the existing service management and orchestration (SMO) arrangement.

II. Related Work

Energy-saving rApps and context. Learning to switch base stations off is established. Li et al.'s TACT framework learns switching policies with a transfer actor-critic controller, using traffic-load states and an explicit cost combining modelled energy consumption and delay [7]. The BeGREEN intelligence plane demonstrated an rApp that switches a capacity cell using a traffic predictor [1], and O-RAN defines an A1 policy type for energy-saving objectives [8]. CAMINO adds weather, road-traffic and rollback context to intent mediation through fixed rules [2]. Our focus is how both source selection and a forecast service's internal configuration adapt to the operator's decisions. Recorded counters support the switching evaluation, but do not establish energy savings or superiority to these controllers.

Hierarchical selection and delayed feedback. Classical work studies hierarchical learning automata [9] and the convergence of teams and hierarchies [10]. Bandit algorithms such as Exp3 learn from the selected action's payoff [11], and delayed-feedback methods adapt such algorithms to late observations [12]. In each, the learner receives the payoff. The method used here shows that a lower selector can instead recover the outcome from the direction of its parent's weight update [3]. We apply that result to an rApp and a shared forecast service with different learning rules, under slot-delayed feedback, and supply no new regret or optimality guarantee.

Adapting forecasts to downstream decisions. Prediction accuracy need not identify the best input to a downstream decision. Smart "Predict, then Optimize" uses the downstream optimisation objective and constraints to train predictions according to their decision error [13]. We address the same distinction between prediction quality and decision usefulness through configuration selection. The simulated products' error characteristics stay fixed while the service's product and alert-threshold weights learn. Its learning selectors don't receive weather labels or the operator's scoring rule, and they don't need a gradient through that rule. The root receives the outcome of the rApp's switching decision, and the weight-change bit carries that outcome down the selected path. Comparing learned configurations with the most accurate fixed setting therefore tests adaptation to the operator, rather than an improvement in forecast accuracy.

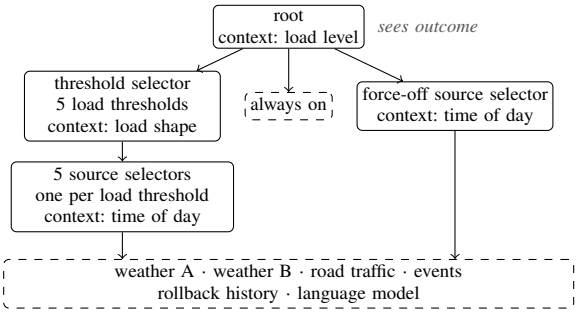


Fig. 1. Base rApp tree (eight selectors). Solid boxes learn. Dashed options do not. The grouped box represents five distinct source selectors. Each of these, and the force-off source selector, chooses among the same six sources. In Section V, weather B becomes the shared learning forecast service in Fig. 3.

Agents in network management. COMA addresses multi-agent credit assignment using a centralised critic, decentralised actors and a counterfactual baseline that varies one agent's action while fixing the others [14]. Chergui et al. study RAN and edge agents built on language models that negotiate configurations and share a memory of contexts, actions and outcomes, while retaining separate objectives [15]. Our selectors update from the same recovered outcome along the chosen path, without estimating individual contributions through a critic or consulting a shared outcome store. The weight-change bit still carries the binary outcome's information. This gives a complementary interface for adapting separately controlled components while their contexts, scoring rules and internal alternatives remain local. A language model can serve as a source without being trained by the feedback at all. The forecast service of Section V sits behind the same source interface but does learn, with its own update rule, from the bit it receives.

III. The Energy-Saving rApp, Its Sources and the Data

A. Tree and Sources

Tree. Fig. 1 shows the base tree. Each solid box is a selector as defined in Section I. The root chooses among three switching strategies, which Fig. 1 shows as its three children: always on, the threshold selector and the force-off source selector. The first keeps the capacity cell on. The second applies a load-threshold

rule. The third proposes a switch-off whatever the load. The root's context is the cell's load in the previous 15-minute slot, classified as low, medium or high. The cutoffs divide the recorded loads across all cells and time slots into three groups with approximately equal numbers of observations. The threshold selector chooses among five thresholds. Its context is the shape of the cell's load over its last four slots, one of three clusters found by k-means on those four loads scaled by their mean. Each threshold, and the force-off branch, has its own source selector, which chooses one of six context sources and uses a six-hour time-of-day band as its context. A proposed switch-off goes ahead only if the chosen source reports no adverse condition. An adverse report, or no response, keeps the cell on. A decision through the threshold branch draws its whole path, including the source, whether or not the threshold permits the switch-off. The report changes the action only when the threshold permits the switch-off, but every selector on the path receives its bit for every decision. The selectors below the root learn from the weight-change bit. In the base experiment the sources return fixed reports and do not learn. Section V replaces weather B with a learning forecast service.

Size. The base tree has eight selectors. With three load levels, three load shapes and four time-of-day bands, it stores at most 30 weight vectors with 168 weights in all, 1,344 bytes in double precision. A decision through the threshold rule updates three vectors and passes two bits. A decision to keep the cell on updates only the root. These figures cover the selection logic, not the cost of querying sources or computing contexts.

Sources. In the base experiment, weather source A detects adverse weather with probability 0.90 and raises a false alarm with probability 0.05. The base version of source B detects with probability 0.80, raises false alarms with probability 0.10 and gives no response in 5% of condition windows. The road-traffic and event sources detect with probabilities 0.85 and 0.95 and raise false alarms with probabilities 0.05 and 0.02. These reports are simulated. The rollback source reports adverse if any condition held in the same window the previous day. The sixth source is a language model (Gemma-4-31B-it, temperature 0). It reads the four simulated reports as text and answers HOLD or PROCEED. Its answers were computed once and cached. The language model is not trained. Each source selector learns how much to trust each source from the weight-change bit, with no labels saying which reports were right.

Conditions. Conditions are simulated on 30-minute windows, on one sequence for the whole region. We call one such 83-day sequence a condition timeline. The experiments use five timelines, drawn with different seeds. Weather is clear or adverse, and adverse spells last six hours on average. Road incidents occur between 06:00 and 22:00. On any day, a three-hour evening event occurs with probability 0.15. Each condition is a reason to keep capacity on, and switching off while one is active counts as wrong. Their effect on traffic is not simulated. Source A works normally for the first half of each timeline. Over the third quarter its detection and false-alarm probabilities move steadily to 0.5, where its reports carry no information. In the last quarter it stops responding. We call these periods healthy, drift and outage.

B. Network Data

We use Dataset_03 of a public release of performance counters from a commercial 5G network [4], [5]. It covers 83 days in 15-minute intervals. We keep the 277 sectors with complete records for an NR 3500 capacity cell (n78, 100 MHz) and an NR 2100 cell (n1, 20 MHz). The capacity cell is the one that may be switched off. We keep its slots with complete volume, utilisation and energy records. One round is one decision, for one capacity cell in one 15-minute slot. Rounds run slot by slot, and within a slot over the cells, so one tree serves all of them. A run has 2,074,760 rounds, a median of 275 per slot.

To judge a switch-off, we add the capacity cell's recorded downlink volume to the NR 2100 cell of the same sector. We convert the added volume to utilisation using the NR 2100 cell's median volume per percentage point of utilisation over the whole dataset. The transfer is feasible if the NR 2100 utilisation would stay at or below 80%. The five thresholds propose a switch-off if, in the previous slot, that utilisation would have stayed at or below 50, 60, 70, 80 or 90%. The load used for contexts is the capacity cell's recorded downlink volume. The counters are a fixed record and no decision changes later traffic. These calculations therefore score each decision against the recorded slot, and do not measure what switching would have done.

Outcome. The right action is to switch off when the transfer is feasible and no condition is active, and to stay on otherwise. The right action gives an outcome of 1, and a wrong switch-off gives 0. A missed switch-off gives 1 with probability c. We call the mean outcome the decision score. The source comparison uses $c = 0.5$, so a missed opportunity costs half as much as a wrong switch-off on average. The forecast service demonstration uses $c = 0.8$, giving a wrong switch-off five times the expected penalty of a missed opportunity (Section V). A deployed rApp would receive richer information, such as network telemetry, operational context and feasibility estimates, and apply the operator's rule to produce a binary judgement. Defining and validating that judgement, including how to assess missed switch-off opportunities, is the rApp's responsibility.

IV. Learning From One Bit

A. Update Rules and the Bit

We summarise the method of [3], which gives the proofs. A selector v chooses among $N_v \geq 2$ children. It maps what it observes to a finite context using a map of its own, and keeps a vector of weights $\mathbf{w}_v^k$ for each context k. The weights are positive, sum to one and start equal. When a decision reaches v, it finds its context k and samples child c with probability $w_{v,c}^k$.

The rApp's rule. The rApp uses proportional reward-penalty (PRP). With step size $0 < \eta < 1$, a positive bit updates the weights as

$$w_c' = (1-\eta)w_c + \eta, \qquad w_j' = (1-\eta)w_j, \quad j \neq c, \quad (1)$$

and a negative bit as

$$w'_c = (1-\eta)w_c, \qquad w'_j = w_j \frac{1 - w_c + \eta w_c}{1 - w_c}, \quad j \neq c. \quad (2)$$

The right-hand sides use the weights from before the update. We use the PRP implementation released with the method [3].

The forecast service's rule. The forecast service uses either PRP or linear reward-ϵ-penalty ($L_{R\epsilon P}$). The latter has the same reward update (1), but on a negative bit uses

$$w'_c = (1-\epsilon\eta)w_c, \quad w'_j = (1-\epsilon\eta)w_j + \frac{\epsilon\eta}{N_v - 1}, \quad j \neq c. \quad (3)$$

We set $\eta = 0.005$ throughout and $\epsilon = 0.01$ inside the forecast service. Vectors are normalised after each update. We don't add a weight floor or an exploration term. The two rules can produce different weights. The method preserves the behaviour of whichever rule each selector uses.

Recovering the outcome. The root uses the decision's outcome $o_t \in \{0, 1\}$ as its bit. Every other selector on the path uses $s = \mathbf{1}[\Delta > 0]$, where Δ is the change in the weight its parent gives it. We call s the decoded bit. The selector then applies its own rule to its own chosen child. Selectors off the path do nothing. Both rules raise the chosen weight on a positive bit and lower it on a negative bit while weights remain interior. Consequently, each decoded bit equals the root outcome. With the same initial weights, random draws and update order, the entire hierarchy makes the same choices and ends with the same weights as a copy given the outcome at every level [3]. This identity does not require the selectors to share a rule or context map, and does not itself imply optimal decisions.

B. Runs and Timing

We use five condition timelines and four selection seeds per timeline, with 2,074,760 rounds per run. Simulated source and product parameters are held fixed across these timelines. The base experiment compares three ways of maintaining source weights. The adaptive rApp updates them throughout. A frozen-source comparator learns identically through the healthy period, then freezes only its source weights at the start of the drift. A uniform-source comparator keeps those weights equal from the start. The root and threshold selectors continue learning in every arm. Three arms for each of the 20 timeline and seed combinations give 60 runs in total. The language model answered 19,667 distinct prompts covering 20,160 condition windows, and every answer parsed.

Timing. Fig. 2 gives the procedure for one slot. All cells in the slot are decided with the weights from the start of the slot. Each selector on a decision's path records its context and the child it chose. Once the slot closes, its outcomes are applied one decision at a time, down each decision's path from the root. Each selector reads the direction of its parent's update for that request, so updates for other cells cannot change it. Contexts stay local and are never passed down the tree. Both experiments use this slot-delayed feedback.

Reproduction. A package accompanies the paper [6]. It holds the processed counter trace, the cached model answers, the seeds and a pinned copy of the update rule. One route checks the saved results against reference hashes, recomputes the reported numbers and redraws the figures. Separate replay commands rerun the source comparison and the experiment with the forecast service, writing their results apart from the references. Both use the cached answers and don't need a model endpoint.

```
Q ← [ ]                                          ▷ paths decided in this slot
for each cell in the slot, in replay order do
    Choose a path with the slot-start weights.
    Append its triples (v, k_v, c_v) to Q as path P.
end for
for each path P in Q, in order, once the slot closes do
    s ← outcome of P's action
    for each (v, k_v, c_v) in P, from root to leaf do
        Save the selected weight and its siblings' sum.
        Update w_v^{k_v} with bit s using v's rule.
        s ← bit decoded from this update                 ▷ next bit
    end for
end for
```

Fig. 2. Procedure for one slot. A triple records a selector v, its context k_v and its chosen child c_v. Every decision in the slot comes before any outcome is applied. The final bit is unused when the selected child is a fixed leaf. The experiment with the forecast service uses the stable reading of the weight change described in Section VI-C, and the base experiment compares the weight directly.

V. Learning Inside a Weather Forecast Service

A. Extending the Hierarchy into a Forecast Service

In a separate experiment, we replace the base weather-B report generator (Section III-A) with a weather forecast service B that has its own hierarchy (Fig. 3). Its first selector chooses one of two forecast products, P1 and P2. Each product has a selector over five alert thresholds: 0.25, 0.75, 1.25, 1.75 and 2.25. A product reports adverse weather if its alert value exceeds the selected threshold. Missing responses retain the rApp's rule of keeping the cell on.

One forecast service instance serves all six source selectors in the rApp. In this replay, its three selectors each use a single context, with their weight vectors shared across callers. They do not use the callers' load, shape or time contexts. Each invocation follows one path, and the parent of B's product selector on that path is the source selector that made the request. The longest path now has five selectors: three in the rApp and two inside the forecast service. A request on that path produces four bits, one crossing the service boundary. There are eleven selectors and at most 33 vectors containing 180 weights in total, or 1,440 bytes in double precision, excluding pending requests, contexts and source data.

Only the root receives the operator's outcome explicitly and knows the scoring rule. Each source selector sends B one bit for the completed request, whether B's weight at that selector rose or fell. B uses it to update its product selection, then passes the direction of that update to the selected product's alert-threshold selector. The rApp need not know which product or threshold B used. The forecast service retains its selected path until the corresponding feedback arrives. In the replay, ordered local path

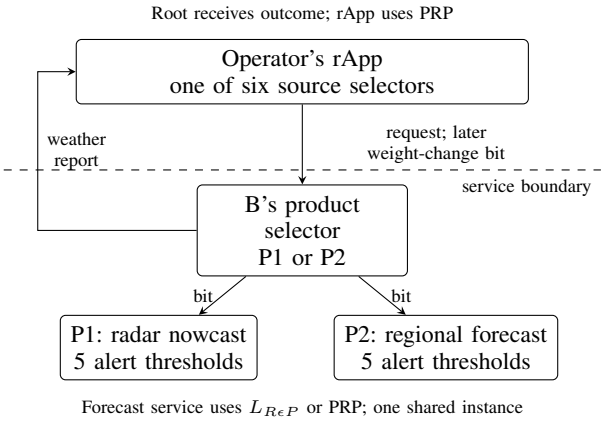


Fig. 3. A forecast service hierarchy behind the rApp's source interface. Only the chosen product's alert-threshold selector is updated. Requests and weather reports are the service interaction, and the weight-change bit is the only feedback passed down. The rApp's scoring rule and contexts, and B's internal choices, remain local. The boundary is modelled in one process.

records supply this association. A deployment would require matching request identifiers and ordered application of the bits. The bit is tied to one request and to the caller's update for it. The caller doesn't pass down the size of the weight change, its context or its scoring rule. The bit carries the outcome's information. The method doesn't remove the need for feedback. It fixes it as one bit derived from the caller's own update, at every level. We assume access to this request-specific bit and do not infer it from changes in aggregate request frequency.

B. Products and the Operator's Objective

Product profiles. These simulated products replace the base B report generator. We call their error characteristics product profiles. Detection and false-alarm rates now depend on product and threshold. The base rates of 0.80 and 0.10 do not apply. P1 represents a radar nowcast, whose alert value has distribution $\mathcal{N}(3,1)$ in adverse weather. In clear weather, 30% of windows have clutter with alert value $\mathcal{N}(2,1)$, and the rest have alert value $\mathcal{N}(0,1)$. P2 represents a regional forecast. It misses 40% of adverse windows, assigning them $\mathcal{N}(0,1)$ alert values, and uses $\mathcal{N}(3,1)$ for the remainder. In clear weather its alert value is $\mathcal{N}(0,1)$. Both products share a 5% no-response schedule. Their alert values, clutter, missed events and missing responses use separate random streams from the existing replay.

B's most accurate fixed setting. B's own quality measure is weather classification accuracy over windows in which it responds. We evaluate all ten product/threshold settings against the simulated weather and take the most accurate as its fixed setting. In every timeline this is P2 at threshold 2.25. This benchmark has the benefit of the complete timeline's weather labels. B's learning selectors receive no such labels. The rApp's other sources remain as before. The language model continues to read the original simulated reports, including the original weather-B feed, so its cached answers do not depend on the learning forecast service's choices.

Prioritising avoidance of wrong switch-offs. Switching off when the remaining capacity or conditions do not permit it risks service, whereas a missed switch-off forgoes a saving. We therefore focus the forecast service demonstration on an operator that penalises the former mistake more heavily. Correct actions score one, wrong switch-offs zero, and missed switch-offs receive a positive outcome with probability $c = 0.8$. The expected penalties are thus 1 and 0.2 respectively. B doesn't receive c, the condition labels or the operator's transfer-feasibility calculation.

C. Comparisons and Analysis

We use the same five condition timelines and four selection seeds, with 2,074,760 rounds per run. B learns using $L_{R\epsilon P}$ or PRP, and the rApp always uses PRP. Each of these 40 learning runs has a copy with the outcome given to every selector and identical random draws. Ten further runs per timeline/seed hold B at each possible setting while allowing the rApp to learn. All use the same slot-delayed feedback and retain source A's drift and outage schedule.

We separate two questions. First, how useful are B's settings on the requests it actually receives? On each request reaching B in a learning run, we score all ten fixed settings with the same upstream decision, condition draws and outcome randomness. The difference from the most accurate fixed setting measures adaptation at the forecast service under this exposure. These counterfactual scores are used only for analysis, never for learning. They include requests where the upstream threshold already rules out a switch-off.

Second, does installing a learning B improve the whole rApp? Here we compare the decision score over all decisions against the separate runs with B fixed. The rApp may learn different source weights in these runs, so this comparison includes changes in the requests reaching B. For both comparisons we average the four selection seeds within each condition timeline, then form paired differences and 95% Student-t intervals across the five timelines. These intervals describe variation in simulated conditions and selection on one recorded trace.

VI. Results

The evaluation checks that the hierarchy supports useful adaptation and preserves the behaviour of explicit outcome feedback. The forecast-setting and source-weight comparisons measure the usefulness of the adaptation. The paired replays check the feedback implementation.

A. Tuning the Forecast Service to the rApp

Table I shows the rApp's decision-score gain on requests served by B. With $L_{R\epsilon P}$, B raises that score by 3.72 percentage points over its most accurate fixed setting. PRP also improves it, by 2.64 points. Both differences are positive in every condition timeline. The forecast service can therefore learn a configuration better suited to this operator's preference for avoiding wrong switch-offs, using only the bit and without knowing the scoring rule.

TABLE I

RAPP DECISION-SCORE GAIN ON B'S REQUESTS OVER THE MOST ACCURATE FIXED SETTING, P2 AT THRESHOLD 2.25, FOR THE OPERATOR WITH $c = 0.8$. COMPARISONS USE THE SAME UPSTREAM CHOICES. PAIRED 95% INTERVALS USE FIVE CONDITION TIMELINES AFTER AVERAGING FOUR SELECTION SEEDS PER TIMELINE.

B's rule	Gain (percentage points)	95% interval
$L_{R\epsilon P}$	+3.72	[+2.22, +5.22]
PRP	+2.64	[+1.13, +4.15]

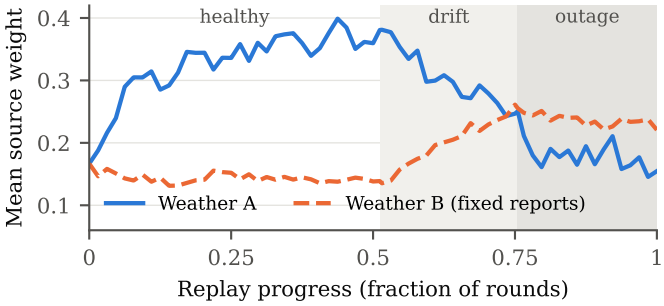


Fig. 4. Source weights in the adaptive arm of the base experiment, with fixed reports, slot-delayed feedback and $c = 0.5$. Curves average equally over instantiated source-selector/time-band vectors within each run, then over 20 runs. Shading marks A's drift and outage, starting at rounds 1,062,005 and 1,566,198. The schedule is defined in time and the number of cells per slot varies, so these starts fall at 0.51 and 0.75 of the rounds. Weights are selection probabilities, not request fractions.

B's product weights show the change behind this result. With $L_{R\epsilon P}$, its mean final weight for P1 ranges from 0.87 to 0.99 across the five timelines, even though weather-classification accuracy favours P2. P1 detects weather events more readily but also raises more false alarms. The operator's scoring rule makes that trade-off useful, and B learns from its consequences without receiving weather labels or the operator's preferences explicitly.

Across all rApp decisions, the score differences are $+0.10$ points with interval $[-0.05, +0.25]$ for $L_{R\epsilon P}$ and $+0.04$ with $[-0.08, +0.17]$ for PRP. These comparisons do not establish a whole-rApp gain. Table I establishes improved usefulness on B's requests under the stated operator objective.

B. Source Trust and Failure Response

The following results use the base experiment, with fixed source reports and $c = 0.5$, to isolate how the rApp responds to source quality. Only the treatment of source weights differs between its adaptive, frozen and uniform arms.

Moving away from a failing source. Fig. 4 shows the average weights of sources A and B. As A drifts, its weight falls from 0.381 at the end of the healthy period to 0.251 at the end of the drift, while B's rises from 0.139 to 0.256. Once A stops responding, choosing it keeps the cell on, and its weight falls further, to 0.155 by the end of the run. No change detector is involved, only the selectors' ordinary updates. This addresses the source-reliability limitation identified in CAMINO [2].

What continued learning adds. Table II compares the outage behaviour. Continuing to adapt raises the score by 7.05 percentage points over freezing the learned source weights, with paired 95% interval $[5.72, 8.37]$, and by 5.72 points over uniform weights, with interval $[5.29, 6.14]$. The same averaging and interval procedure as in the comparison with the forecast service is used. Adaptive and frozen runs have identical decisions and weights throughout the healthy period. Their later difference therefore measures the benefit of continuing to learn which sources to use. Keeping alternative sources available does not by itself account for the gain.

TABLE II

OUTAGE BEHAVIOUR WITH FIXED SOURCE REPORTS AND $c = 0.5$. ENTRIES ARE MEAN PERCENTAGES OVER THE OUTAGE PERIOD ACROSS 20 RUNS. FROZEN SOURCE WEIGHTS RETAIN THEIR VALUES AT THE END OF THE HEALTHY PERIOD. UNIFORM WEIGHTS ARE HELD EQUAL THROUGHOUT. ROOT AND THRESHOLD CHOICES CONTINUE LEARNING IN EVERY ARM.

Metric (%)	Adaptive	Frozen	Uniform
Decision score	85.75	78.71	80.03
Switch-offs / decisions	43.14	27.17	47.23
Wrong switch-offs / decisions	3.49	2.87	8.67
Wrong / switch-offs	8.14	10.82	18.43

Compared with frozen weights, adaptation makes more switch-offs and has fewer errors among them, while increasing wrong switch-offs as a fraction of all decisions. The score captures this trade-off. Compared with uniform weights, adaptation improves the score and reduces both error rates.

C. Learning From the Bit Reproduces Outcome Feedback

The experiment with the forecast service checks paths of up to five selectors, with different context maps and different learning rules on either side of the boundary. All 20 pairs with $L_{R\epsilon P}$ inside B and all 20 with PRP have identical choice sequences, final weights and recorded metrics. None of the 117,773,717 decoded bits differs from the root outcome. Learning from the bit therefore gives exactly the choices that explicit outcome feedback would have given, with slot-delayed feedback throughout. The source comparison also records zero decoding mismatches.

Reading the weight change in floating point. A weight close to one can fail to rise in floating point even when the exact update is positive. The experiment with the forecast service therefore reads the direction from the selected weight when that weight is at most one half. Otherwise it reads the direction from the sum of the siblings' weights, which falls on a positive bit, summed directly with `math.fsum`. The parent computes the bit locally and exposes no sibling weight to its child. Every replay checks the decoded bits against the outcomes.

VII. DISCUSSION AND LIMITS

What the application demonstrates. The implementation connects the rApp's internal selectors and the weather forecast service through the same one-bit interface. The rApp retains its scoring rule and local contexts, while the service retains its product hierarchy and chooses its own learning rule. Paired replays show that this arrangement preserves explicit-outcome

learning when outcomes arrive after the decision interval closes. The source and forecast comparisons show useful adaptation under the stated operator objectives. Together, these results demonstrate a concrete network-management application of the hierarchical method in [3].

Interface assumption. The service boundary is modelled within one process. The replay retains each selected path, reads weight changes through local state access and applies feedback in order after the interval closes. A deployment would need to associate each bit with the retained request and preserve the update order. We assume this interface can be provided; no commercial API restriction is established. The bit carries the binary outcome's information. The paired replays test preservation of learning across this interface, while the source and forecast comparisons test the usefulness of adaptation. They do not establish superiority over other adaptive controllers.

Scope of the evaluation. Traffic comes from recorded counters, but the conditions, product profiles and transfer calculation are modelled. Product profiles have different error characteristics. The results do not estimate the performance of actual forecast products. No result measures energy, since a recorded trace cannot show what switching off would have saved. Feasibility assumes the NR 2100 cell takes all the traffic, an 80% utilisation ceiling and a volume-to-utilisation ratio fitted on the whole dataset. Load levels also use the whole trace, and load shapes are clustered on its first week, which the test includes. All runs share one recorded network trace. Counterfactual product comparisons use information available to the evaluator only. The language model is used without task-specific training. The forecast service result comes from learning product and threshold choices.

Next steps. A testbed trial could implement the one-bit interface between separately controlled components, let actions change the network and compute outcomes from telemetry. Measuring energy would require both the switched cell and the cells taking its traffic. The BeGREEN intelligence plane, demonstrated on emulated cells, offers such a setting [1]. Its traffic predictor could also be evaluated by its effect on switching decisions, allowing a prediction service's model choice to learn below the operator's switching policy.

VIII. Conclusion

This paper implements an energy-saving rApp as a modular hierarchy that learns from one bit of feedback at each level. Only the root receives the outcome of the rApp's completed switching decision explicitly. Each lower selector recovers it from the direction of its own weight change and updates its local choices. A simulated weather forecast service illustrates a component with its own internal hierarchy, tuning its product and alert threshold without receiving the rApp's scoring rule. Replays of 83 days of counters from 277 capacity cells demonstrate adaptation away from a failing source and tuning of forecast responses to an operator prioritising avoidance of wrong switch-offs. Paired replays reproduce the decisions and learned weights of explicit outcome feedback despite delayed updates, different local contexts and different learning rules. The implementation thus applies the hierarchical method across both internal rApp components and a modelled service boundary. The evaluation uses simulated conditions and forecast products. A testbed implementing the one-bit interface between separately controlled components is the next step.

Acknowledgment

This work is supported by the Smart Networks and Services Joint Undertaking (SNS JU) under the European Union's Horizon Europe research and innovation programme, project FLECON-6G (101192462). Views and opinions expressed are however those of the author(s) only and do not necessarily reflect those of the European Union or SNS-JU. Neither the European Union nor the granting authority can be held responsible for them.

References

[1] M. Catalan-Cid, D. Reiss, G. Castellanos, and J. Armstrong, "Demo: BeGREEN intelligence plane for AI-driven energy efficient O-RAN management," in *Proc. IEEE INFOCOM Workshops*, 2025.

[2] J. Armstrong, E. Fallon, and S. Fallon, "CAMINO: Contextually aware mediation of intent for network orchestration," *Telecommunication Systems*, vol. 88, 2025, art. no. 96.

[3] J. Armstrong, "Hierarchical learning automata without a shared reinforcement signal: Recovering the environment response from action-probability changes," arXiv:2605.00921, https://doi.org/10.48550/arXiv.2605.00921, 2026.

[4] P. Lehoczký, M. Turcsány, L. Krajčovičová, F. Zatroch, M. Kajan, and M. Galinski, "Performance management counters from live 5G, 4G and 2G radio access network," *Scientific Data*, 2026.

[5] ——, "Performance management counters from live 5G, 4G and 2G radio access network," Zenodo, version 1, https://doi.org/10.5281/zenodo.17815388, 2026.

[6] J. Armstrong, "Reproduction package for 'Learning From One Bit of Feedback in a Modular Hierarchical rApp'," Zenodo, https://doi.org/10.5281/zenodo.23356827, 2026.

[7] R. Li, Z. Zhao, X. Chen, J. Palicot, and H. Zhang, "TACT: A transfer actor-critic learning framework for energy saving in cellular radio access networks," *IEEE Transactions on Wireless Communications*, vol. 13, no. 4, pp. 2000–2011, 2014.

[8] O-RAN Alliance, "A1 interface: Type definitions," O-RAN.WG2.A1TD-R004-v09.00, ETSI TS 103 988 V9.0.0, 2025.

[9] M. A. L. Thathachar and K. R. Ramakrishnan, "A hierarchical system of learning automata," *IEEE Transactions on Systems, Man, and Cybernetics*, vol. SMC-11, no. 3, pp. 236–241, 1981.

[10] M. A. L. Thathachar and V. V. Phansalkar, "Convergence of teams and hierarchies of learning automata in connectionist systems," *IEEE Transactions on Systems, Man, and Cybernetics*, vol. 25, no. 11, pp. 1459–1469, 1995.

[11] P. Auer, N. Cesa-Bianchi, Y. Freund, and R. E. Schapire, "The nonstochastic multiarmed bandit problem," *SIAM Journal on Computing*, vol. 32, no. 1, pp. 48–77, 2002.

[12] P. Joulani, A. György, and C. Szepesvári, "Online learning under delayed feedback," in *Proc. 30th International Conference on Machine Learning*, ser. Proceedings of Machine Learning Research, vol. 28, no. 3, 2013, pp. 1453–1461.

[13] A. N. Elmachtoub and P. Grigas, "Smart "predict, then optimize"," *Management Science*, vol. 68, no. 1, pp. 9–26, 2022.

[14] J. Foerster, G. Farquhar, T. Afouras, N. Nardelli, and S. Whiteson, "Counterfactual multi-agent policy gradients," in *Proc. AAAI Conference on Artificial Intelligence*, vol. 32, no. 1, 2018, pp. 2974–2982.

[15] H. Chergui, F. Rezazadeh, M. Catalan Cid, P. Sayyad Khodashenas, D. Camps Mur, and C. Verikoukis, "Toward an unbiased collective memory for efficient LLM-based agentic 6G cross-domain management," arXiv:2509.26200v2, 2026.