Provable Novice-Learning System — Hardened Blueprint

Provable Novice-Learning System — Hardened Blueprint

====================================================

Below is a hardened, auditable blueprint that (a) structurally prevents a small open‑source LLM “student” from using pretraining to answer; (b) forces learning from session‑local NOTES/OER; and (c) produces verifiable proofs of teaching without relying on human data.

## 0) Threat model & target guarantee

Threat model. The “student” is a small OS LLM (≤7B) with broad pretraining. We treat it as an adversary that will exploit any recognizable domain cues or world knowledge.

Target guarantee. Before seeing session NOTES/OER, the student’s accuracy on session items is provably near chance. After exposure, accuracy rises only when the student can cite session evidence. All improvements are logged and verifiable from artifacts alone.

## 1) Make prior knowledge useless (information gating > behavior prompts)

### 1.1 Codebook anonymization (mandatory, keyed, pervasive)

* Mapping: Sample a per‑session seed s → derive a bijective mapping M\_s: DomainTerm → CodeToken with no morphological hints (e.g., amygdala → NARO‑4, photosynthesis → ZAB‑17).
* Scope: Apply M\_s to everything the student sees: stems, options, feedback, NOTES, OER. Never reveal unmapped domain terms to the student.
* Rotation: Use multi‑key: s\_def for definitions, s\_fig for figures/tables, s\_app for application passages. Reveal cross‑key joins only through NOTES drills (prevents simple memorization from one surface form).
* Numeric scrambler: Random invertible affine transform per session for numerics/units: x\_real → x' = a·x + b and unit\_real → U' via a unit codebook. Only NOTES contain (a,b, U↔U'). Many items depend on decoding, so pretraining arithmetic gives no advantage.

### 1.2 Relation obfuscation + counterfactualization

* Generate a synthetic micro‑ontology with randomized labels but preserved structure (e.g., a 20‑node DAG of causes/effects or definitions/examples). Tutor content explains these relationships only with codes.
* When lifting real domains, shuffle and relabel relations (e.g., “A inhibits B” → “NARO‑4 offsets ZAB‑17”), but keep internal consistency. Pretraining cannot map back to the original domain.

### 1.3 Leakage tripwires (control items)

* Include control probes that require a mapping not yet revealed (e.g., querying NARO‑4 before its flashcard). Correct early answers indicate leakage; track and exclude those items from mastery—and flag the run.

## 2) Evidence‑bound answering (JSON‑only, auditable)

### 2.1 Output contract (machine‑checkable)

Student must return:

{

"chosen\_index": 2,

"citations": [

{"doc":"OER\_03","start":582,"end":649},

{"doc":"NOTES","line":17}

],

"confidence": 0.62

}

* Grammar‑constrained decoding enforces this schema.
* Coverage score (e.g., BM25/overlap) is computed between cited spans and the keyed rationale for the correct option. If coverage < τ, the attempt is scored as a guess (0 credit), even if the index is correct. This prevents lucky picks and forces evidence use.

### 2.2 Witness check (derivability test)

A witness verifier re‑answers the item using only the cited spans, via a deterministic algorithmic rule (string match / entailment with thresholds). The student earns credit only if the witness also picks the same index → you obtain a mechanized proof that the supplied evidence suffices.

## 3) Students: no‑knowledge and low‑knowledge baselines

### 3.1 Algorithmic student (no LLM)

* Starts at chance using NOTES‑to‑option similarity (BM25/TF‑IDF/token overlap) + simple tie‑breakers (position priors, overlap with definitions previously learned).
* Improves only as NOTES accumulate. This is your floor and a leakage sentinel.

### 3.2 Closed‑book LLM student (small OS model)

* Input = anonymized question + NOTES (+small OER snippet window); never raw domain terms.
* Constrained to JSON schema; must cite.
* A small linear/logistic head trained online on features from NOTES (e.g., overlap scores, page ranks) can blend with the LLM score—still auditable and pretraining‑free because the head sees only NOTES features.

Diagnostic only: An open‑book mode (OER+NOTES) should also be near chance before reveal of mappings; if not, you’ve leaked.

## 4) “Proofs of teaching” — what you can prove after a run

All proofs rely on artifacts that a third party can re‑compute from seeds.

### 4.1 Proof‑of‑Exclusion (pretraining is useless)

* Protocol: Evaluate the same anonymized test with (i) the small LLM without NOTES/OER; (ii) the algorithmic student without NOTES/OER.
* Claim: Accuracy ≈ 1/K (K options).
* Test: Binomial test & 95% CI vs chance. Register seeds. If > chance, you have leakage to fix.

### 4.2 Proof‑of‑Use (answers derive from session evidence)

* Re‑score each answered item using the witness verifier on only the cited spans.
* Metric: % items where witness = student (target ≥ 90% for credited items).
* Counterfactual deletion: Remove cited spans from OER/NOTES and re‑run the student; accuracy should drop back toward chance. This is a direct, reproducible falsification test.

### 4.3 Proof‑of‑Growth (learning curve)

* Design: Randomly interleave teach and test items per mapping.
* Model: logit(p\_correct) = α\_skill + β\_skill \* exposures\_skill + γ \* coverage + δ \* (time) with hierarchical shrinkage over skills.
* Proof: Posterior β\_skill > 0 for most skills; global slope > 0 with ROPE excluding 0. Provide pre‑registered priors and seeds.

### 4.4 Proof‑of‑Transfer

* After definitions are learned, introduce applications that require combining ≥2 mappings. Show performance tracks the min(exposures) or AND‑logic predicted by the ontology.

### 4.5 Proof‑of‑Retention

* Re‑test earlier mappings after a delay (or after NOTES window truncation). Show performance correlates with spaced review count rather than absolute time.

## 5) Integrity & audit: “Proof‑of‑Study (PoSt) ledger”

* Seed log: Record session seed(s), anonymizer keys, numeric transforms.
* Content hashes: SHA‑256 of each OER chunk, item, NOTES snapshot.
* Attestations: For each answer, store (item\_hash, chosen\_index, citations, coverage, witness\_result, timestamp) and hash the tuple.
* Repro script: A one‑command script that, given seeds + ledger, reconstructs all OER/NOTES/items and verifies every witness check and statistic. This is your portable proof pack.

## 6) Item & curriculum design that requires session knowledge

### 6.1 Control & diagnostic items

* Pre‑mapping controls: Use unrevealed codes; expect chance.
* Anti‑leak distractors: Options that are high LM‑likelihood completions under generic English (measured offline) but incorrect in the code world—so prior fluency misleads.
* Sham OER: Insert short, irrelevant passages. The evidence‑bound scorer should penalize citing sham content.

### 6.2 Shaping progression

1) Definition ↔ code flashcards to teach the mapping.

2) Near‑transfer: paraphrases still using single mapping.

3) Compositional: 2–3 mappings required; plausible distractors share 1 of the needed mappings.

4) Application: numeric scrambler + relation lookup necessary.

## 7) Scoring & telemetry (measurable signals)

* Primary: accuracy (credit only if coverage ≥ τ and witness passes).
* Coverage: e.g., BM25 on cited spans vs gold rationale; normalized to [0,1].
* Mapping mastery: for each code term c, track exposures(c), correct(c), and a posterior of p(c) from a Beta‑Binomial.
* Error patterns: confusion matrix over codes; co‑mistake clustering to identify missing links.
* Time/coverage trends: slope of (coverage, latency) over session blocks.
* Integrity flags: % correct on pre‑mapping controls (should be ≈ chance).

## 8) Validation plan (end‑to‑end)

1) Baseline run (n≈200 items): algorithmic student only → start ≈ chance; slope positive as NOTES grow.

2) Leakage check: closed‑book LLM with no NOTES on anonymized items → near chance.

3) A/B anonymized vs. non‑anonymized: same items, un‑mapped vs. mapped. Expect a large accuracy gap pre‑NOTES.

4) Ablation: truncate NOTES window → performance drops specifically on low‑exposure mappings.

5) Randomization inference: shuffle labels; pipeline should return chance and low coverage.

## 9) Implementation blueprint (practical)

### 9.1 Components

* Anonymizer layer
* Inputs: domain text (if any), item drafts, OER.
* Outputs: mapped text using M\_s, numeric scrambler (a,b), unit map.
* Store {seed, M\_s, (a,b), unit\_map} in user stats.
* NOTES pipeline
* Append, after each item: (Q\_mapped, CorrectOption\_mapped, Rationale\_mapped, MinimalWitnessSpans)
* Sliding window (e.g., last 200 lines) to simulate memory.
* Coverage scorer + witness verifier
* Coverage = max(BM25/overlap) between citations and gold spans.
* Witness = deterministic picker using only cited spans (e.g., match key code tokens and relation markers).
* Students
* Algorithmic: BM25/overlap ranking of options vs NOTES.
* Closed‑book LLM: JSON‑constrained; pre‑scored by a small trained head on NOTES features.
* Metrics logger & PoSt ledger
* Hash all artifacts; export a single “proof pack.”

### 9.2 JSON I/O contracts

Item (to student)

{

"item\_id": "S1-I083",

"stem": "When ZAB-17 offsets NARO-4, which outcome is expected?",

"options": ["A: ...", "B: ...", "C: ...", "D: ...", "E: ..."],

"notes\_excerpt": "… (windowed NOTES/OER, mapped) …"

}

Answer (from student)

{

"chosen\_index": 2,

"citations": [{"doc":"OER\_03","start":582,"end":649},{"doc":"NOTES","line":17}],

"confidence": 0.62

}

Scoring response (to logs)

{

"item\_id":"S1-I083",

"correct\_index": 3,

"student\_index": 3,

"coverage": 0.78,

"witness\_pass": true,

"latency\_ms": 2140,

"controls": {"pre\_mapping": false}

}

### 9.3 Pseudocode highlights

Coverage + witness

def score\_answer(item, answer, gold\_spans, tau=0.5):

cov = max\_bm25\_overlap(answer.citations, gold\_spans)

if cov < tau: return 0, cov, False

witness\_idx = pick\_with\_witness\_only(answer.citations, item.options)

ok = (witness\_idx == answer.chosen\_index)

return (1 if ok and answer.chosen\_index==item.correct\_index else 0), cov, ok

Algorithmic student

def algo\_student(item, notes):

scores = [bm25(notes, opt) + token\_overlap(notes, opt) for opt in item.options]

return argmax(scores)

## 10) What makes this provable in practice

* No‑peek guarantee: The student never sees unmapped domain terms or unscrambled numbers → pretraining priors cannot align.
* Control‑validated: Unrevealed‑mapping items remain at chance (live test for leakage).
* Citation‑verified: Credit is awarded only when the student supplies spans that a separate algorithm can use to reach the same answer.
* Seeded reproducibility: Anyone with the seeds can regenerate items/OER/NOTES and re‑verify all outcomes.
* Statistically attested: Learning slopes and transfer are demonstrated with registered models and permutation tests.

## 11) Quick wins (1–3 days)

1) Ship algorithmic student + coverage + witness scoring (credit only if coverage≥τ & witness pass).

2) Simple anonymizer for ~50 high‑signal terms + numeric (a,b) scrambler + unit map; expose as flashcards in NOTES.

3) PoSt ledger with seeds & content hashes; CLI script to re‑verify a session’s results.

## 12) Stretch enhancements

* Progressive reveal drills: gated unlock of codewords; automatic scheduling (spaced repetition).
* Per‑skill micro‑texts in mapped vocabulary (short OER chunks the student must cite).
* Tiny trainable head: logistic on NOTES features for calibrated probabilities and better curves (still auditable).
* Anti‑memorization keys: rotate (a,b) and partial sub‑mappings mid‑session; require re‑consolidation.
* LLM‑likelihood traps: auto‑generate distractors that are highest general LM likelihood but false under the code world.

Bottom line

By constraining information (not behavior), requiring citations, and verifying derivability with a witness algorithm, you get a system where a small LLM cannot coast on pretraining, and any score gains are linked—provably—to what was actually taught in‑session. The PoSt ledger and seeded regeneration make the whole process auditable and reproducible without human annotators.