PROMPT REGRESSION CHECKS

Running Regression Checks on Prompt Changes



AI & DATA

Running Regression Checks on Prompt Changes

Learning outcomes

- Compare two prompt versions against a frozen ten-case suite.
- Identify two improvements and two regressions.
- Recommend a change decision using case-level evidence.
- Audit exact-string oracles from complete raw record packets.
- Validate required case/prompt/trial keys before scoring.
- Compute transition membership, pass rates and reconciliation identities.
- Separate execution states, repeated observations and distinct case coverage.
- Design concrete neighboring tests and a bounded repair/retest decision without changing the frozen target.

Original teaching material. All organizations, people, records and operational rules in exercises are synthetic. Use only the supplied non-sensitive inputs. This is a practice course, not an accredited or regulated certification.

Study the foundations, work through the applied units, complete the practice tasks, and then attempt the assessment. Use only the supplied fictional practice data. Keep your original inputs and your revised work separate.

Core concepts

A regression is loss of previously passing behavior

A revised prompt can solve one problem while breaking another. Regression testing reruns a fixed set of earlier cases to detect this loss. Keep the input, expected behavior and scoring rule unchanged between versions. Version labels alone are not enough; store the actual prompt text or a stable reference. A test that changes along with the prompt cannot establish whether behavior regressed, because the target has moved. The provided ten-case suite uses exact expected values to make the comparison unambiguous.

Classify transitions, not just totals

Each case has four possible transitions: fail to pass is improvement; pass to fail is regression; pass to pass is retained success; fail to fail is retained failure. Count all four and list the case IDs. Equal total pass rates can conceal a trade: two improvements and two regressions cancel numerically while affecting different requirements. A release decision should therefore inspect transition categories, not only the net score. Persistent failures also deserve attention even though they are not newly introduced.

Keep release criteria separate from score arithmetic

For this exercise the proposed version cannot be approved if any previously passing case regresses. This is a conservative local acceptance rule, not a universal software policy. Under that rule, improvements do not buy permission to break earlier behavior. A decision memo should acknowledge the benefits, name the blockers and propose a repair plus rerun. If an organization intentionally changes expected behavior, that requires an explicit specification change and a new versioned test, not relabeling an inconvenient failure after seeing it.

Use a failure to strengthen the suite

After investigating a regression, add neighboring cases that test the same mechanism, but do not replace the original failing case. Preserve the original suite so the old behavior remains visible. A successful retest should include the repaired cases and the full set because the repair itself may have side effects. Record not run separately from fail; absent evidence must not be converted into a passing status. The result is a controlled change history rather than a series of anecdotal prompt improvements.

Worked example

Demonstration: case X passes in v1 and fails in v2, while case Y fails in v1 and passes in v2. Both versions pass one of two, but the change includes one regression and one improvement. An unchanged 50% total does not mean behavior stayed unchanged.

Foundation practice

Prompt v1: Extract the value after the record label. Return UNKNOWN if absent.

Prompt v2: Extract the final value after the record label. Preserve exact spelling.

Rule: Exact expected string is required. No regression from a previously passing case is permitted for approval.

Suite

id	expected	v1	v2
R1	VALUE-1	UNKNOWN	VALUE-1
R2	VALUE-2	UNKNOWN	VALUE-2
R3	VALUE-3	VALUE-3	UNKNOWN
R4	VALUE-4	VALUE-4	UNKNOWN
R5	VALUE-5	VALUE-5	VALUE-5
R6	VALUE-6	VALUE-6	VALUE-6
R7	VALUE-7	VALUE-7	VALUE-7
R8	VALUE-8	VALUE-8	VALUE-8

id	expected	v1	v2
R9	VALUE-9	UNKNOWN	UNKNOWN
R10	VALUE-10	VALUE-10	VALUE-10

Foundation tasks

Guided preparation

Instructions

Read the complete input pack. Create the output columns specified in the capstone, keep the original IDs, and state the unit of work. Complete the first two records manually before using an optional AI assistant. Record any assumption that could change the result.

Independent capstone

Instructions

Create a ten-row regression matrix with v1/v2 pass states and transition class. Identify both improvements and both regressions. Write an approval or hold recommendation and a specific retest plan.

Peer verification and revision

Instructions

Exchange only your student work with a peer. Ask the peer to trace two ordinary results and every exception back to the supplied input. Record the exact item corrected, the original value, revised value and reason. Submit the revised artifact plus a short explanation of the most important remaining limitation.

01 Define regression as a transition under a stable target

A prompt change is often motivated by a visible failure: a missing value, a repeated label or an unwanted format. Fixing that example is useful, but it does not establish that the change is safe for previously working inputs. Regression checking asks whether the revised prompt loses behavior that passed under the same specification. The comparison is about transitions on matched cases, not whether the new wording looks more polished.

Four transitions are possible when both outputs were actually obtained and scored. Fail to pass is improvement. Pass to fail is regression. Pass to pass is retained success. Fail to fail is retained failure. A retained failure is not newly caused by the change, but it remains a real limitation. The four categories partition a complete paired suite; missing output evidence belongs outside that four-way correctness classification.

The foundation comparison has ten cases with exact expected strings and supplied outputs. Its two improvements and two regressions cancel in total pass count, leaving seven passes for each version. That arithmetic does not mean the behavior is unchanged. Different records

now fail. A decision needs the case IDs and their requirements, especially when the local rule prohibits any loss of previously passing behavior.

The expanded worked case concerns equipment-label extraction for a fictional workshop. Each raw record may contain a code label, repeated assignments or distracting note text. The required behavior is fully specified in a contract, and the supplied output snapshots are manually authored teaching fixtures. They are not results of a live AI service. You can reproduce the scoring locally without a paid account, a network call or any claim that a model was actually run.

Keep the target stable across versions: raw input, expected result and scoring semantics. Store the actual prompt text, not just the name v2. If the task specification changes, document that as a separate versioned change rather than quietly altering expected answers after seeing inconvenient failures. A regression claim is meaningful only relative to the target that both versions were supposed to satisfy.

The aim is a defensible change decision. You will build a complete transition ledger, explain specific regressions, distinguish incomplete evidence from incorrect output and design a repair retest. The local no-regression rule determines change eligibility in this course; it does not claim that a small suite proves universal production readiness.

02 Separate specification, prompt, oracle and observation

Four artifacts play different roles. The specification states what the task should do. The prompt is one instruction intended to produce that behavior. The oracle is the expected result or procedure used to judge an output. The observation is the output obtained in a particular run or supplied as a fixture. Confusing these roles makes tests circular: a candidate output should not become its own expected answer.

In the workshop case, label-contract-1 is the specification. It says to use the final matching asset_code line, trim only outer ASCII spaces and preserve the remaining value exactly. The prompts in prompts.json are candidates intended to implement that behavior. The suite contains raw input and expected strings derived from the contract. The output file contains v1 and v2 snapshots that may or may not match those strings.

A prompt can omit part of the specification without changing the intended target. The workshop v2 wording says clean up formatting but does not explicitly restate the absent-value sentinel or exact label matching. That omission is a plausible source of failures, but the supplied outputs do not prove the causal mechanism. Treat prompt inspection as hypothesis generation, and the paired fixtures as evidence of observed behavior under the declared teaching scenario.

An oracle also needs review. If the expected value incorrectly chooses the first label while the contract says final label, the test would reward the wrong behavior. Derive a few expected values manually from the raw source before trusting the entire table. For T02, OLD appears first and NEW last; the oracle is NEW. For T09, the last matching assignment is empty, so the oracle is UNKNOWN, not the earlier OLD.

The foundation matrix supports transition analysis but does not include the original raw record inputs. It therefore cannot establish how those historical expected strings were derived or reproduce a live extraction run. Do not invent raw records and label them as the original evidence. The new workshop and dispatch suites provide separate complete raw packets so the oracle and scoring can be checked end to end within the exercise.

A strong report identifies each artifact explicitly. “Prompt v2 passed seven cases” is weaker than naming the prompt text revision, suite revision, scoring contract and output snapshot file. Reproducibility does not require pretending all provenance is available. It requires stating what is available, preserving it and limiting claims to what that evidence supports.

E02-X1 Identify the four artifact roles

Use contract.json, prompts.json, workshop_suite.json and workshop_outputs.json. Explain which provides specification, candidate instructions, oracle plus raw input and observed fixture output.

Submit: Four-role map and one limitation of the foundation matrix.

03 Make exact-match semantics explicit before scoring

Exact match sounds simple until different readers normalize strings differently. The workshop contract uses Unicode-string equality with no case folding, whitespace collapse or Unicode normalization. A returned value must match the expected string exactly. The task's permitted preprocessing occurs when deriving the selected source value: remove only ASCII spaces at its beginning and end. That does not authorize trimming or repairing candidate outputs before scoring.

The label is case-sensitive and must start at the first character of a line, followed immediately by a colon. asset_code: matches. ASSET_CODE: does not. note: asset_code: FAKE does not match because the label is embedded in a note rather than at the start. These distinctions are local task rules, not claims that every extraction system should behave this way. Their purpose is to make the expected behavior reproducible.

The final matching assignment wins, even if it is empty. If no matching line exists, or if the final selected value becomes empty after the declared outer-space trim, the required output is UNKNOWN. The sentinel is a literal string, not JSON null, an empty string or N/A. This difference matters because T03's candidate v2 output is empty, while its expected result is UNKNOWN.

Preserve internal content. Leading zeroes in 0007 are part of the identifier, not a numeric count to convert. Lowercase ab-007 is not interchangeable with uppercase AB-007. A B contains an internal space that cannot be replaced with a hyphen or deleted. café-9 includes an accent that the contract preserves. Each of these cases tests a distinct way a cleanup instruction can damage an identifier.

Do not quietly normalize only one version's outputs. If v2 receives a whitespace repair before scoring while v1 is judged raw, the comparison changes the scoring rule along with the prompt. If the intended production interface truly allows equivalent forms, define that

equivalence before running the comparison and apply it to both. In this exercise, exact strings are the target, so post-hoc cleanup would hide the behavior being tested.

Visible rendering can make differences hard to see. Use a quoted or escaped representation when inspecting empty strings, trailing spaces or repeated spaces. The raw value remains the scoring object; the escaped display is a diagnostic aid. A report should explain the meaningful character difference rather than dismissing it as cosmetic when the contract explicitly requires preservation.

04 Derive expected outputs from the raw source packet

Read the raw record line by line and apply the contract in order. First identify matching label lines. Next select the final matching line. Then remove only outer ASCII spaces from the value after the first label colon. Finally, use UNKNOWN if the selected result is empty or no label matched. Do not select the last word in the record or the last nonempty value unless the specification says so.

T07 demonstrates why matching comes before selection. Its first line is a note containing asset_code: FAKE, while its second line begins with asset_code: REAL. Only the second line qualifies, so REAL is expected. T12 contains the embedded note but no true label line; its expected value is UNKNOWN. A substring search that grabs anything after asset_code: would give a plausible but unsupported value for T12.

T09 demonstrates why the final assignment can override an earlier usable value. The first line supplies OLD and the second matching line contains only spaces. After trimming, the final value is empty, so UNKNOWN is required. Choosing OLD because it is the last nonempty assignment would implement a different specification. A regression check should expose that difference rather than call both answers reasonable.

For the dispatch capstone, the required label changes to dispatch_code, but the same contract rules apply. D09 contains dispatch_code: Q:09. Only the first colon separates label from value; Q:09 remains the identifier. Splitting on every colon and taking the last fragment would incorrectly return 09. This case tests delimiter handling rather than missingness or label precedence.

An independently derived oracle can be recorded with a short evidence note: “T02, second matching label line, value NEW.” You do not need to copy the whole record into every report. The stable ID and selected line are enough to locate the source. For an absent label, state which matching rule failed rather than cite a nonexistent selected line.

Before scoring all outputs, verify ordinary, boundary and exception cases manually. One correct ordinary label does not establish that the oracle handles empty final assignments or embedded decoys. A small set of deliberate checks can reveal an incorrect reference procedure before it produces a large, internally consistent but wrong score table.

E04-X1 Audit five oracle values

Derive expected strings for T02, T03, T08, T09 and T12 from workshop_suite.json raw inputs. Cite the selection or absence rule for each.

Submit: Five exact strings with evidence notes.

ID	Raw input (newline shown as \n)	Expected
T01	asset_code: AX-1	AX-1
T02	asset_code: OLD\nasset_code: NEW	NEW
T03	note: no code supplied	UNKNOWN
T04	asset_code: ab-007	ab-007
T05	asset_code: Q-2	Q-2
T06	asset_code: 0007	0007
T07	note: asset_code: FAKE\nasset_code: REAL	REAL
T08	ASSET_CODE: ZZ	UNKNOWN
T09	asset_code: OLD\nasset_code:	UNKNOWN
T10	asset_code: café-9	café-9
T11	asset_code: A B	A B
T12	note: asset_code: FAKE	UNKNOWN

Exact source spellings for whitespace-sensitive records are shown below in monospace. Quotes delimit the strings, and the two-character escape \n represents one line break. Keep the spaces inside the quotes; the JSON assets preserve the original strings.

```
T05_input = 'asset_code:   Q-2  '
T09_input = 'asset_code: OLD\nasset_code:   '
D10_input = 'dispatch_code: A  B'
```

05 Freeze a suite without freezing learning

A frozen suite preserves a comparison target. It does not mean the team can never add cases or correct a genuinely wrong specification. It means the v1/v2 comparison refers to a specific set of raw inputs, expected strings and scoring rules that are not changed opportunistically after seeing results. Version the suite when you intentionally change its membership or oracle.

Store the exact prompt text alongside its version label. A filename v2 can be overwritten, and two people can use the same label for different wording. The supplied manifest includes SHA-256 digests of the prompt strings so accidental text changes can be detected. A digest

identifies bytes; it does not prove that the prompt is good, that a model ran it or that the associated output is authentic.

Keep run configuration separate from prompt content. A real comparison might record model version, parameters, tool availability and input preprocessing. Changing any of those can affect outputs and confound attribution to the prompt. The teaching fixtures deliberately do not claim a provider, model or temperature because no live service was used. Recording unknown or not applicable is more honest than filling plausible configuration values.

The expected key set is part of the freeze. For workshop-1, twelve case IDs, two prompt versions and one trial require twenty-four output records. The pairing key is case_id, prompt and trial. The same number of rows is not enough if one key is duplicated and another is missing. A complete matrix contains exactly one valid observation for every required key and no unexpected keys.

Preserve the source packet and output snapshot separately. If a corrected output replaces an earlier failed result without a new run or fixture version, the comparison history disappears. Keep the original evidence and record the new candidate or retest explicitly. A test suite should make changes easier to understand, not encourage editing history until it looks green.

When adding neighboring cases, assign new stable IDs and document the mechanism they test. Do not replace the original failing case with an easier example. The original demonstrates a known failure under a known target. New cases broaden coverage; they should not erase the evidence that motivated the repair.

06 Validate matrix integrity before calculating pass rates

Scoring begins only after the observation matrix is structurally sound. Check required case IDs, prompt versions and trial IDs. Reject duplicate keys, identify missing keys and report unexpected keys. Otherwise, a row-count total can look correct while the comparison silently pairs the wrong observations or counts one easy case twice.

The deliberately broken incomplete_outputs.json illustrates this problem. It removes T03/v2/trial 1, duplicates T01/v1/trial 1 and adds T99/v1/trial 1. The file still contains many plausible outputs, but it is not the required workshop matrix. Do not compute an ordinary paired pass-rate comparison from it. Report the exact key defects and request a corrected observation set.

Missing and not_run are related but distinct evidence states. A missing key means no record exists for a required observation. A record with status not_run explicitly says the observation was not attempted or not supplied. Neither establishes output correctness. A record with status error records an execution problem rather than a valid generated answer. The four transition categories apply only when both completed outputs can be scored.

Do not turn an error into the string UNKNOWN unless the model actually returned that value as a completed answer. UNKNOWN is the task's semantic sentinel for absent source information. A transport failure, parser failure or skipped test is an execution state. Conflating the two can make a failed run appear to pass an absent-label case by accident.

Use exact pairing keys rather than row order. Sorting v1 rows alphabetically and v2 rows by time can create false regressions if you compare positions. Join by case ID and trial, then compare each version against the same expected string. Preserve the output value and status for diagnosis even when the final matrix stores only pass flags and transition labels.

Integrity failures are blockers to interpretation, not necessarily defects in either prompt. A duplicated row may come from an export or collection mistake. Fix the evidence pipeline before attributing the problem to model behavior. A trustworthy report distinguishes dataset-integrity issues, execution issues and task-output failures so each can be addressed by the appropriate action.

E06-X1 Audit the broken observation file

Compare incomplete_outputs.json with the workshop manifest: 12 cases, v1/v2, trial 1. List duplicate, missing and unexpected keys, then decide whether an ordinary paired result is valid.

Submit: Three exact key lists and one decision.

07 Build the four-way transition ledger

For each valid pair, compare v1 output with the expected string and do the same for v2. The two Boolean results determine the transition. Do not compare the two outputs only with each other: two identical wrong answers are a retained failure, not a retained success. The oracle is the reference for both versions.

T02 has expected NEW, v1 OLD and v2 NEW. Its states are fail then pass, so it is an improvement. T03 has expected UNKNOWN, v1 UNKNOWN and v2 an empty string. Its states are pass then fail, so it is a regression. The difference between empty and UNKNOWN is required by the contract, not a subjective preference about output style.

T01 matches AX-1 in both versions and is a retained success. T11 expects A B, while v1 returns A-B and v2 returns AB. Both fail, so it is a retained failure even though the erroneous outputs differ. The transition label says whether the requirement was met, not whether the same wrong characters were produced.

A useful ledger includes case ID, slice, expected string, both raw outputs, both pass flags and transition. The slice identifies a behavior mechanism such as final empty assignment or internal spaces. This helps a reviewer see whether improvements and regressions cluster around a particular prompt change. Keep the raw strings available so the label can be independently verified.

The workshop matrix contains four improvements, four regressions, three retained successes and one retained failure. Those counts sum to twelve. Both versions pass seven cases: retained successes plus improvements for v2, retained successes plus regressions for v1. The identical pass totals conceal substantial churn in which requirements are satisfied.

Do not interpret transition counts as severity weights. Four formatting-sensitive identifier regressions may have different operational consequences from four missing optional comments in another task. This course's exact-string contract and no-regression change rule

make the decision unambiguous. In a different workflow, severity and release criteria would need to be defined explicitly before the comparison, not inferred from a convenient aggregate.

E07-X1 Score four representative transitions

Score T01, T02, T03 and T11 using the raw output snapshots and exact expected strings.

Submit: Four rows with v1_pass, v2_pass and transition.

08 Reconcile totals, rates and net change

Aggregate arithmetic is useful when its denominator is clear. For the twelve-case workshop suite, v1 passes seven of twelve and v2 also passes seven of twelve, approximately 58.33 percent each. The net pass-count change is zero. The transition ledger shows why: four improvements add passes and four regressions remove passes. Retained successes contribute to both totals; retained failures contribute to neither.

A simple reconciliation catches many spreadsheet mistakes. v2 passes minus v1 passes must equal improvements minus regressions for a complete single-trial binary matrix. Here seven minus seven equals four minus four. The four transition counts must sum to the number of cases: four plus four plus three plus one equals twelve. If either identity fails, inspect pairing, missing rows or category formulas before interpreting the result.

Do not call zero net change zero behavioral change. Eight of twelve cases change pass state, while four retain their correctness state. The changed-state proportion is eight divided by twelve, about 66.67 percent. That statistic describes churn in pass status, not overall output similarity or risk. T11's raw output changes even though its pass state remains fail, so state churn and textual change are different measures.

Percentage points and relative percent change also differ. If a pass rate rose from fifty to sixty percent, the increase would be ten percentage points and twenty percent relative to the original fifty. In the workshop example both rates are equal, so both changes are zero. Keep the raw counts beside percentages to avoid rounding hiding a small but important regression in a small suite.

The capstone has a different case count and transition pattern. Do not carry the workshop denominator or totals into it. Calculate from its ten rows and verify the same identities. A reliable report names the suite and trial scope beside every aggregate so a reader does not mistake a worked example's result for the independent assessment.

Finally, a pass rate does not itself decide release. Under the local rule, any regression blocks approval even if the candidate's total rises. Score arithmetic describes outcomes; the acceptance policy interprets them. Keeping those steps separate prevents a positive average from silently overriding the explicit protection for previously passing behavior.

ID	v1 pass	v2 pass	Transition
T01	True	True	retained_success

ID	v1 pass	v2 pass	Transition
T02	False	True	improvement
T03	True	False	regression
T04	True	False	regression
T05	True	True	retained_success
T06	False	True	improvement
T07	True	True	retained_success
T08	True	False	regression
T09	False	True	improvement
T10	False	True	improvement
T11	False	False	retained_failure
T12	True	False	regression

There is another useful reconciliation at the row level. A complete matrix with twelve cases and two prompts contains twenty-four output strings, but it contains only twelve paired transitions. If you divide the number of improvements by twenty-four, you are using an output denominator for a case-transition question. For the workshop, the improvement share is four of twelve, while the candidate output pass rate is seven of twelve. These are different measures with the same denominator here only because each case has one output per candidate.

A transition can also be summarized as a signed change: improvement is plus one, regression is minus one and both retained categories are zero. The workshop signed changes sum to zero. This representation is useful for arithmetic checks, but it deliberately loses the distinction between retained success and retained failure. A zero row must therefore keep its original transition label so a persistent defect does not disappear into the same bucket as a stable pass.

The capstone illustrates why net change alone is insufficient even when it is negative. A decline of one pass could arise from one regression and no improvements, or from three improvements and four regressions. The operational repair work differs greatly. The dispatch packet has the latter pattern. Naming the four lost behaviors and three gains provides a much better repair agenda than merely saying performance fell by one.

When presenting rounded rates, preserve exact fractions in the evidence record. Seven of twelve is not exactly fifty-eight percent, and six of ten is exactly sixty percent. Rounding should make a report readable without changing the underlying gate, which operates on case membership. A small numerical display change caused by rounding must never turn a regression into a retained success.

09 Inspect slices and mechanisms without inventing causality

A slice groups cases by a meaningful input feature. The workshop uses slices such as repeated_label, absent, label_case, final_empty and internal_space. These labels help connect a failure to a requirement. They are not demographic groups or a statistical guarantee. With one case in most slices, the evidence is a specific example, not a reliable estimate of a broad population failure rate.

The v2 improvements include selecting the final repeated value, preserving leading zeroes, honoring the final empty assignment and preserving an accent. Its regressions include the absent sentinel, value capitalization, label-case matching and embedded decoy rejection. A clean-up instruction may be related to some of these errors, but the output snapshot alone does not prove why the model behaved that way.

Write failure hypotheses as hypotheses. “The cleanup wording may encourage case normalization” is bounded. “The model always uppercases identifiers because of this sentence” is not supported by one fixture. To investigate causality, a controlled follow-up would change the suspected instruction while holding other factors stable and rerun the relevant and full suites. The current exercise supplies fixed snapshots rather than executing that experiment.

Use counterexamples to sharpen the diagnosis. T07 and T12 both contain an embedded asset_code phrase, but only T07 also has a valid label line. v2 passes T07 and fails T12. This suggests that testing only decoys alongside a valid label would miss the no-valid-label boundary. Add a neighbor with a different decoy value and no real label, while retaining T12 as the demonstrated failure.

Avoid overgeneralizing from a slice name. T04 tests case preservation in the value; T08 tests case sensitivity in the label. Both involve capitalization but protect different contract rules. A repair that lowercases the entire record could worsen both. The ledger should identify the exact relationship that failed rather than treat all case-related errors as one interchangeable problem.

Slice analysis helps prioritize repair work and select neighbors, but it should not replace the full regression check. A targeted fix for missing labels can accidentally alter repeated-label behavior. The full frozen suite remains the guard against side effects outside the narrow mechanism currently under investigation.

E09-X1 Write a bounded failure hypothesis

Compare T07 and T12. Explain what their difference reveals and propose one new raw neighbor case with an exact expected result. Do not claim a proven model cause.

Submit: Contrast explanation, hypothesis and concrete neighbor.

10 Treat repeated trials as repeated evidence, not new cases

A model can produce different outputs for the same input under repeated execution. A real evaluation may therefore repeat trials, but trial identity must be explicit. Three outputs for T01 are not three different input cases. They provide repeated observations of one case under a declared configuration. Counting them as independent coverage exaggerates the variety of behavior tested.

The repeated_outputs.json file is a separate fictional illustration for T01, T02 and T11 with three trials per prompt. T01 passes all three v1 trials and two of three v2 trials. T02 passes one of three v1 trials and all three v2 trials. T11 passes none in either version. Across nine trial observations per prompt, v1 has four passes and v2 has five.

Those totals do not erase the T01 loss. If the local repeated-trial protection were that a previously all-pass case must remain all-pass, T01 would block the change. That is a separate illustrative criterion from the main one-trial workshop gate. State which rule you are applying and do not blend the repeated illustration into the main twelve-case pass rate.

Pairing trials requires care. If trial labels correspond to matched conditions, compare matching case and trial keys. If trials are merely independent samples without matched randomness, do not imply that trial one in v1 and trial one in v2 share every stochastic condition. The teaching data use fixed numbered observations for arithmetic practice, not a claim of controlled random seeds in a live service.

Report case coverage and observation count separately. The repeated file covers three cases and eighteen outputs total: three cases times two prompts times three trials. It does not cover eighteen cases. A summary that says “tested eighteen examples” would be misleading if readers infer eighteen distinct inputs. Include per-case pass counts so a fragile formerly reliable case remains visible.

Repeated trials also cannot cure a bad oracle. If the expected value is wrong, every repeated score will repeat the same reference error. Check the contract and raw input first, then assess variation. More observations are useful only when their identity, target and collection conditions are understood.

11 Distinguish not run, execution error and task failure

An output can fail the task only when there is an output to evaluate under the scoring rule. If the execution was skipped, record not_run. If the request failed before producing a valid response, record error. If a completed response is the wrong string, record a task failure. These categories support different remedies: scheduling or collection, execution troubleshooting, and prompt or system repair.

Consider three supplied mini-records for an absent-label input whose expected value is UNKNOWN. Record A has status completed and output UNKNOWN; it passes. Record B has status completed and output an empty string; it fails the task. Record C has status not_run

and output null; it is missing test evidence, not a pass or a semantic UNKNOWN. The same absent source information does not make an absent execution acceptable.

A transport error may be operationally important enough to block a release, but keep its label accurate. Calling it a regression requires an explicit policy about comparing execution reliability, not pretending it is an incorrect extraction result. A workflow can have separate gates for completeness, execution success and semantic correctness. The course's main change gate requires complete completed observations before a normal paired decision.

Do not silently drop failed requests from the denominator. Reporting only successful completions can make a brittle candidate look better because difficult or failed executions disappear. Report the required observation count, completed count, error count and not-run count. Then state which correctness statistics are valid on which subset and whether the release gate remains blocked by missing evidence.

An interrupted rerun is not equivalent to a failed candidate. Preserve the completed observations and mark the missing ones. The next step may be to resume or repeat the missing work under consistent conditions, not to rewrite the prompt. Conversely, do not turn an incomplete run into a provisional approval simply because all observed outputs look good. Absence of a known failure is weaker than completion of the required evidence.

A good checker refuses to produce a normal approved result from incomplete data. It can still provide diagnostics that help repair the evidence set. This makes the tool useful without laundering uncertainty into a confident pass rate.

E11-X1 Classify execution and correctness

Expected output is UNKNOWN. A={status: completed, output: UNKNOWN}; B={status: completed, output: empty string}; C={status: not_run, output: null}; D={status: error, output: null}.

Submit: Four classifications and a release implication.

12 Implement a small, inspectable snapshot scorer

A small scorer should do a few things transparently: load the suite and observations, validate unique keys, confirm complete required membership, score exact strings and classify transitions. It should not call an AI service, repair candidate outputs or overwrite fixtures. Keeping those responsibilities separate makes the result easier to trust and the failure easier to diagnose.

The supplied score_snapshots.py accepts a suite JSON path and an output JSON path. It derives expected keys for v1, v2 and trial 1 from the current suite, reports duplicates, missing keys and unexpected keys, and refuses the normal matrix when observations are incomplete or noncompleted. For a complete matrix it prints every case's pass flags and transition, then aggregate counts and the local decision.

The script uses explicit exceptions or diagnostics for input problems rather than relying on assertions as application validation. Assertions are useful in verification tests but can be disabled by optimized Python execution. A teaching utility that validates external data should

preserve its validation behavior regardless of that setting. The separate verification script checks the utility with positive and negative fixtures.

The scoring comparison is output == expected. No strip, lower or replacement is applied to candidate output. This is intentional: the contract requires exact strings. A learner who adds .strip() to make a trailing-space output pass has changed the scorer, not fixed the prompt. If the contract later permits that normalization, version the scorer and re-evaluate both candidates under the revised rule.

Inspect the tool's output before trusting the headline. A table of case-level transitions should reconcile to totals, and the listed regressions should drive the no-regression decision. If a tool prints a higher pass rate but omits the regression IDs, it is insufficient for this task. The most useful automation makes the evidence easier to inspect rather than replacing it with a single opaque score.

Run the scorer from any working directory by passing the appropriate file paths. The utility does not depend on the author's workspace or a special checks directory name. Portability matters because a reviewer should be able to reproduce the same result from a copied course folder without modifying the instructional source.

Complete runnable snapshot scorer

```
#!/usr/bin/env python3
"""Score fixed snapshots without calling a model or changing source files."""
from pathlib import Path
from collections import Counter
import json
import sys


def score(suite, observations):
    cases = suite['cases']
    if not cases:
        raise ValueError('Suite must contain at least one case')
    case_ids = [case['id'] for case in cases]
    if len(case_ids) != len(set(case_ids)):
        raise ValueError('Duplicate case ID in suite')
    if any(not isinstance(case['expected'], str) for case in cases):
        raise ValueError('Expected values must be strings')
    expected_keys = {
        (case_id, prompt, '1')
        for case_id in case_ids
        for prompt in ('v1', 'v2')
    }
    keys = [
        (row['case_id'], row['prompt'], row['trial'])
        for row in observations
    ]
    counts = Counter(keys)
    duplicates = sorted(key for key, n in counts.items() if n > 1)
    missing = sorted(expected_keys - set(keys))
    unexpected = sorted(set(keys) - expected_keys)
    invalid = [
        key for key, row in zip(keys, observations)
        if row['status'] != 'completed'
        or not isinstance(row['output'], str)
    ]
    problems = {
        'duplicates': duplicates,
        'missing': missing,
        'unexpected': unexpected,
        'noncompleted_or_invalid': invalid,
    }
    if any(problems.values()):
        return {'integrity': problems, 'decision': 'INCOMPLETE_EVIDENCE'}
    index = {key: row for key, row in zip(keys, observations)}
    labels = {
        (False, True): 'improvement',
        (True, False): 'regression',
        (True, True): 'retained_success',
        (False, False): 'retained_failure',
    }
    rows = []
    for case in cases:
        a = index[(case['id'], 'v1', '1')]['output']
        b = index[(case['id'], 'v2', '1')]['output']
        pa = a == case['expected']
        pb = b == case['expected']
```

```
        rows.append({
            'id': case['id'],
            'expected': case['expected'],
            'v1': a,
            'v2': b,
            'v1_pass': pa,
            'v2_pass': pb,
            'transition': labels[(pa, pb)],
        })
    groups = {
        label: [row['id'] for row in rows
                if row['transition'] == label]
        for label in labels.values()
    }
    a_count = sum(row['v1_pass'] for row in rows)
    b_count = sum(row['v2_pass'] for row in rows)
    decision = ('HOLD' if groups['regression']
                else 'ELIGIBLE_UNDER_LOCAL_CHANGE_GATE')
    return {
        'integrity': problems,
        'rows': rows,
        'transitions': groups,
        'pass_counts': {'v1': a_count, 'v2': b_count},
        'required_cases': len(cases),
        'net_change': b_count - a_count,
        'decision': decision,
    }


def main():
    if len(sys.argv) != 3:
        raise SystemExit('Usage: score_snapshots.py SUITE.json OUTPUTS.json')
    suite = json.loads(Path(sys.argv[1]).read_text(encoding='utf-8'))
    outputs = json.loads(Path(sys.argv[2]).read_text(encoding='utf-8'))
    result = score(suite, outputs)
    print(json.dumps(result, indent=2, ensure_ascii=False))
    return 2 if result['decision'] == 'INCOMPLETE_EVIDENCE' else 0


if __name__ == '__main__':
    raise SystemExit(main())
```

13 Write a change decision that follows the declared gate

The workshop candidate has four improvements and four regressions. Both prompts pass seven of twelve cases, but the local rule permits no loss of previously passing behavior. The decision is hold v2. Name the blockers T03, T04, T08 and T12, acknowledge improvements T02, T06, T09 and T10, and retain T11 as a persistent failure needing attention.

A useful decision does not erase the benefits. The candidate handles the final repeated assignment, leading zeroes, final empty assignment and accent in these fixtures better than v1. Those gains can inform a repair. But improvements do not buy permission to violate the

declared no-regression rule. The decision should explain the policy relation rather than imply that v2 is worthless or that v1 is universally good.

Distinguish change eligibility from production readiness. A later candidate could eliminate every regression while leaving T11 failing in both versions. Under the narrow no-regression change gate, it might become eligible if the required evidence is complete. That does not establish that the workflow meets all production quality requirements. Persistent failures must remain visible, and any additional release standard must be stated separately.

Do not change the rule after seeing the result. If an organization decides that a specific regression is an intentional specification change, it needs an explicit authorized decision and a new versioned target. The comparison should then preserve the old result and explain the new requirement. Simply relabeling a failed expected value to match v2 would invalidate the original regression claim.

The memo should name the evidence scope: fixed fictional workshop snapshots, twelve raw cases, exact-string scoring, one trial per prompt and no live-service execution. It should recommend a repair and full rerun rather than claim the revised wording is already verified. A proposed next prompt is a hypothesis until its outputs are collected and evaluated.

A concise result can still be complete: “Hold v2: 7/12 passes for both, four improvements offset by four regressions; the no-regression gate fails. Repair the missing-sentinel, value-case, exact-label-case and decoy-only behavior; retain T11 in the suite and rerun all twelve plus new neighbors.” Every clause in that summary is supported by a row or declared rule.

E13-X1 Draft the workshop decision

Use the complete workshop ledger. Report pass counts, all four transition ID lists, the no-regression decision and a specific retest scope.

Submit: A compact evidence-based decision record.

14 Repair the prompt without erasing the failure evidence

A prompt repair should target the observed requirement gaps while retaining behaviors that already work. For the workshop, a reasonable candidate instruction would explicitly state exact line-start label matching, final assignment selection, UNKNOWN for absent or empty selected values and preservation of internal characters. That wording addresses hypotheses suggested by the fixtures, but the course does not claim a live model will obey it.

Keep the old prompts and output snapshots unchanged. Store the repair as a new prompt version, such as v3-proposed, and identify which failures motivated it. A comparison between v1 and v2 remains part of the evidence history even if a later version performs better. Overwriting v2 with the repair would make its old output file appear to come from text that never produced it.

A retest plan should name both affected cases and the full suite. T03 checks absent-label sentinel behavior, T04 value case, T08 exact label case and T12 decoy-only input. T11

remains a persistent internal-space failure. The full twelve-case suite protects the four gains and retained successes from side effects. New neighbors broaden coverage but do not replace any existing case.

Include concrete neighbors rather than “test more edge cases.” For a label-case neighbor, use Asset_code: MIX and expect UNKNOWN under the exact lowercase label rule. For an internal-space neighbor, use asset_code: A C and expect A C with all three internal spaces. For final-empty behavior, use a valid earlier code followed by asset_code: and expect UNKNOWN. Each has a raw input and an oracle before any new output is collected.

If the repair changes another system component, record it. A new preprocessing step that lowercases labels or normalizes Unicode is not merely a prompt edit. It may change the task contract and invalidate comparisons under the old exact-match rule. Test changes in a controlled way so the team can distinguish prompt effects from pipeline effects.

Do not announce success from inspecting the new prompt alone. The next evidence needed is a complete output matrix under the frozen contract and recorded configuration, followed by the same scoring and gate checks. Until then the status is proposed repair awaiting test, not fixed. This distinction protects the user from treating an attractive instruction as demonstrated behavior.

15 Design neighboring cases and preserve their purpose

Neighboring cases should probe the mechanism behind a failure while changing enough surface detail to reduce memorization. T12 uses an embedded label in a note with no true label line. A neighbor can change note to memo and FAKE to SHADOW while retaining the no-valid-label structure. If the revised prompt simply memorized T12's exact text, that neighbor could expose the weakness.

Vary one relevant factor at a time where possible. A neighbor that changes label capitalization, adds a second assignment, inserts accents and removes the final newline all at once may fail, but it will be hard to diagnose. Controlled variation makes the test informative. Broader combination cases can be added later to test interactions after the basic mechanism is understood.

Positive neighbors are useful too. A case with a decoy note followed by a valid exact label should still return the valid value. A case with ordinary lowercase label and lowercase value should preserve the value. If a repair blocks all records containing the word note, it might fix T12 while breaking T07. Testing only negative examples can reward an overly restrictive workaround.

Every new case needs a stable ID, raw input, expected result, slice and reason for inclusion. The reason should name the protected requirement, not merely say edge case. A small, well-explained suite can be more useful than a large collection of near-duplicates whose purpose is unclear. Coverage is about meaningful distinctions, not the number of rows alone.

Keep development cases separate from a genuinely held-out assessment when you need evidence of generalization. Once a case is used to tune a prompt, it becomes development

feedback. It can remain valuable for regression protection, but passing it no longer provides the same independence as an unseen test. The dispatch capstone is independent learner work in this course, not a claim about a real model's unseen evaluation.

A neighbor plan is complete when a reviewer can reproduce the inputs and oracle without asking what you meant. Do not leave blank worksheets or a request to invent examples later. Supply the concrete records, explain the mechanism, and state that their model outputs are not yet available unless the packet actually includes them.

E15-X1 Supply a concrete three-case retest extension

Create N02 for mixed-case label rejection, N03 for preserving three internal spaces, and N04 for final-empty override. Give exact raw text and expected strings.

Submit: Three complete case records, not just topic names.

A specification correction needs its own evidence trail. Suppose a reviewer discovers that a suite's expected value for a repeated label uses the first assignment even though the approved contract says final assignment. Correcting that oracle is not gaming the score if the discrepancy is documented and the same corrected target is applied to both candidates. Preserve the original test result, record why the oracle was wrong and rerun the comparison under a new suite revision. The corrected score answers a different, better-specified question; it should not be presented as though the old file never existed.

Contrast that with changing UNKNOWN to an empty string only because v2 returned empty. Unless an authorized specification change now permits the empty representation, that edit simply rewards the observed failure. The evidence must lead the target correction, not the desire for a green result. A reviewer should be able to cite the contract clause or authorized requirement that justifies every oracle change.

A similar discipline applies to deleting cases. Removing an obsolete requirement can be legitimate after a documented specification change. Removing T12 because it is difficult is not. If a case is excluded from a new suite, retain its ID, exclusion reason and the version in which the requirement changed. Otherwise, a rising pass rate can reflect shrinking coverage rather than improved behavior.

16 Independent capstone: dispatch-code prompt change

The dispatch capstone uses ten fictional dispatch records, a different label and new identifier values. Read dispatch_suite.json and dispatch_outputs.json under label-contract-1. The required label is dispatch_code. The exact prompt text for dispatch_v1 and dispatch_v2 is supplied in prompts.json; the output file uses v1 and v2 as the comparison labels within this separate suite.

Begin by auditing the oracle from raw input. D02 has repeated assignments. D03 has no label. D05 has an empty assignment. D06 has a label phrase embedded in memo text. D09 includes an internal colon, and D10 includes two internal spaces. These are distinct

requirements, so do not assume that a failure in one category predicts the others without checking their actual strings.

Validate the required matrix of ten cases, two prompts and one trial. Then produce a ten-row transition ledger with expected value, both raw outputs, pass flags and transition class. Report all four transition ID lists, both pass counts and rates, net change and the reconciliation between improvement-minus-regression and candidate-minus-baseline passes.

Apply the same local no-regression change gate. Identify every newly failing case and the exact protected behavior it loses. A candidate can improve some difficult records and still fail the gate. Do not treat an unchanged persistent accent failure as a new regression, and do not hide it simply because it does not affect the no-regression count.

Your retest proposal must include the full ten-case suite plus two concrete new neighbors for the internal-colon and double-space behavior. Supply their raw inputs and exact expected strings, but mark their outputs not run. A plan is not evidence of a successful retest. State what additional observation set would be needed to reconsider the change decision.

The final memo should be bounded to this fixture comparison. No live AI call, production deployment or repair execution is part of the supplied evidence. You are establishing the result of a reproducible snapshot evaluation and a justified next step, not predicting how an unspecified model will behave on every dispatch record.

E16-X1 Evaluate the complete dispatch change

Audit raw oracles, validate matrix integrity, build all ten transition rows, calculate counts/rates and decide under the no-regression rule. Supply two concrete unrun neighbors and a full retest plan.

Submit: Complete ledger, decision memo and neighbor inputs.

17 Handoff a reproducible regression record

A reproducible handoff includes the contract version, suite ID, raw inputs, expected values, exact prompt texts, output snapshots, scoring method and decision. It also includes any missing provenance or unperformed work. Another reviewer should be able to derive the same row-level outcomes from the files without relying on your memory of which prompt was intended to win.

Keep the decision separate from the score file. A score report can show counts and transitions; the decision memo explains how the declared gate applies and what remains unresolved. This separation makes it possible to revise a release policy transparently without rewriting the historical observations. It also prevents a dashboard's default green color from becoming an unexamined authorization.

Record limitations that matter. The fixture outputs are fictional and fixed. The exact-match scorer does not assess broad semantic quality beyond the declared extraction contract. The suite covers the listed cases, not every possible Unicode string or malformed record. No model nondeterminism, latency, cost or production integration was measured. These

boundaries do not make the exercise useless; they define precisely what its result establishes.

If a real team later runs the prompts, preserve the collection metadata and distinguish those observations from the teaching snapshots. Do not append live outputs into the fixture file without a new run identifier. The comparison should identify whether changes arose from prompt text, model configuration, input preprocessing or test membership. Stable records make diagnosis possible when several things change close together.

The final habit is to distrust convenient summaries until they reconcile with cases. Equal totals can hide regressions, a higher total can violate a protected-behavior gate, and a missing output can vanish from a denominator. Read the ledger, check the oracle and inspect the evidence state. Then write a decision that matches the rule rather than the hoped-for outcome.

A good regression process does not prevent change. It makes change accountable: gains remain visible, losses are named, repairs are tested and uncertainty is not disguised as success. The result is a controlled learning loop in which prompt revisions earn confidence through reproducible evidence instead of persuasive wording alone.

Assessment

Choose one best answer per question. Questions 1–10 check the foundation concepts; questions 11–20 apply the extended study units. Complete and submit the practical tasks separately. A strong quiz score does not replace correct practical work.

1. A case passes under v1 and fails under v2 against the same oracle. What is it?

A. Retained success.

B. Retained failure.

C. Regression.

D. Improvement.

2. A case fails under v1 and passes under v2 against the same oracle. What is it?

A. Retained failure.

B. Regression.

C. Improvement.

D. Missing evidence.

3. Both versions pass seven cases. What follows?

A. Exact matching guarantees no behavior changed.

B. They pass exactly the same cases.

C. A no-regression gate is automatically satisfied.

D. Their totals may hide offsetting improvements and regressions.

4. Which must stay fixed for a matched correctness comparison under the same declared target?

A. Inputs, expected behavior and scoring semantics.

B. Only the percentage display format.

C. Only the prompt filename.

D. Only the number of rows.

5. When may an expected value be changed legitimately?

A. Whenever v2 produces a plausible alternative.

B. After an explicit justified specification correction or change, versioned separately.

C. Whenever it raises the pass rate.

D. When the output is shorter than the oracle.

6. T11 has two different wrong outputs. How is its transition classified?

A. Retained failure because both miss the oracle.

B. Regression because the strings differ.

C. Retained success because both are nonempty.

D. Improvement because the text changed.

7. Why keep T12 after adding a decoy-only neighbor?

A. It lets the team avoid a full-suite rerun.

B. The new case replaces the need for the old source.

C. It guarantees every future model will fail.

D. It preserves the demonstrated failure as a regression check.

8. Under the local no-regression rule, do four improvements offset four regressions?

A. No; any lost previously passing case blocks the change.

B. Yes if the new prompt is shorter.

C. Yes if the total pass rate ties.

D. Only if regressions are rounded out of the rate.

9. After a targeted prompt repair, what retest is required here?

A. Only fresh easy cases.

B. Only the case that motivated the edit.

C. No outputs if the wording appears complete.

D. Affected cases and the full frozen suite, plus declared neighbors.

10. A required observation has status not_run. How should it be treated?

A. Missing test evidence, not a correctness result.

B. A pass if its expected result is UNKNOWN.

C. A completed wrong extraction by definition.

D. A retained success inherited from v1.

11. T09 ends with an empty exact label assignment after OLD. What is expected?

A. OLD, because it is the last nonempty value.

B. OLD followed by UNKNOWN, to preserve both values.

C. UNKNOWN, because final empty assignment invokes the sentinel.

D. An empty string, because the final line is empty.

12. Which line matches the workshop label rule?

A. Exact lowercase label at the start: asset_code: X

B. Embedded label: note: asset_code: X

C. Uppercase label at the start: ASSET_CODE: X

D. One leading ASCII space before asset_code: X

13. Why must candidate output not be stripped before exact scoring?

A. All whitespace is always meaningful in every task.

B. That would change the declared output-comparison rule.

C. Python cannot compare strings containing spaces.

D. The source input must never be trimmed under any rule.

14. A file has one duplicate key and one missing key but the expected row count. What should happen?

A. Score by position because the counts match.

B. Report key-integrity defects before normal paired scoring.

C. Delete any row until a percentage looks reasonable.

D. Treat the duplicate as the missing case.

15. The workshop has four improvements and four regressions. What is the net pass-count change?

A. 4.

B. -4.

C. 0.

D. 8.

16. Repeated fixtures cover three cases, two prompts and three trials. How many distinct cases are covered?

A. 9.

B. 6.

C. 3.

D. 18.

17. What does a SHA-256 prompt digest establish here?

A. Proof that the model executed the prompt.

B. Permission to change the suite without versioning.

C. Proof that every expected value is correct.

D. An identifier for the exact prompt bytes.

18. D09 source is dispatch_code: Q:09. What is the expected value?

A. "Q"

B. "Q:09"

C. "09"

D. "UNKNOWN"

19. Which claim is justified from T07 passing and T12 failing in v2?

A. The prompt change has proven universal causality.

B. A decoy-only fallback is a hypothesis worth testing with neighbors.

C. The oracle should accept FAKE for T12.

D. The model always fails every record with notes.

20. A repaired prompt is written but no new outputs are supplied. What is its status?

A. Equivalent to a completed full-suite pass.

B. Approved because the author expects improvement.

C. Verified fix because every rule is mentioned.

D. Proposed repair awaiting a complete retest.

Sources and further reading

NIST AI Risk Management Framework
https://airc.nist.gov/airmf-resources/airmf/

Python unittest documentation (3.15 page as retrieved)
https://docs.python.org/3/library/unittest.html

NIST AI RMF Playbook: Measure
https://airc.nist.gov/airmf-resources/playbook/measure/

Python language reference: assert statement
https://docs.python.org/3/reference/simple_stmts.html#the-assert-statement