[1] Prerequisite and eligibility rules

Prerequisite skill blocking

```
IF mastery(skill_prereq, level) < prereq_threshold(level)
THEN
  block(module)
  recommend(prerequisite_modules)
  set_mode("BLOCK_AND_REMEDIATE")
```

Missing critical foundation

```
IF skill_is_foundational(skill)
AND mastery(skill, level) < minimum_required
THEN
  restrict(exploration)
  recommend(foundational_remediation)
  set_mode("BLOCK_AND_REMEDIATE")
```

Level jump restriction

```
IF target_level > current_level + 1
AND mastery(current_level) < promotion_threshold
THEN
  block(level_jump)
  recommend(level_gap_modules)
```

[2] Assessment validity rules

Insufficient evidence

```
IF assessment_attempts(skill) < min_attempts
THEN
  prevent_mastery_update()
  require(additional_attempt)
```

Invalid assessment behaviour

```
IF completion_time < min_time_threshold
AND score_high
THEN
  flag("guessing_behavior")
```

```
  reduce_confidence_weight()
```

[3] Failure detection and recovery rules

Repeated conceptual failure

```
IF same_concept_error_count >= N
AND no_improvement_trend
THEN
  recommend(concept_remediation)
  downgrade_progression_priority()
  set_mode("RECOVERY")
```

Difficulty mismatch

```
IF time_spent >> expected_time
AND score_low
THEN
  recommend(easier_variant OR prerequisite_review)
  set_mode("RECOVERY")
```

Stagnation detection

```
IF attempts > max_attempts
AND mastery_change < epsilon
THEN
  pause_progression()
  inject(alternative_explanation)
```

[4] Retry and attempt weighting rules

Attempt weight assessment

```
FOR each attempt i:
  weight_i = increasing_weight(i)
```

Mastery update rule

```
mastery = weighted_average(score_i * weight_i)
```

[5] Mastery evaluation rules

Skill-level threshold check

```
IF mastery(skill, level) >= threshold(level)
THEN
  mark_skill("MASTERED")
ELSE
  mark_skill("IN_PROGRESS")
```

Improvement-based promotion

```
IF improvement_rate >= improvement_threshold
AND latest_score >= near_pass
THEN
  allow_progression()
```

[6] Mastery decay rules

Inactivity decay

```
IF time_since_last_activity > decay_window
THEN
  reduce_confidence(mastery, decay_factor)
```

Review trigger

```
IF confidence < confidence_threshold
THEN
  recommend(review_module)
```

[7] Progression rules

Normal progression unlock

```
IF skill_status == "MASTERED"
AND prerequisites_satisfied
THEN
  unlock(next_modules)
  set_mode("PROGRESS")
```

Topic continuity

```
IF learner_focus_topic == current_topic
THEN
  prioritize(related_modules)
```

[8] Exploration rules

Safe exploration eligibility

```
IF no_blocking_rules_active
AND recovery_not_needed
THEN
  allow_exploration()
```

Alternative path suggestion

```
IF mastery_high
AND engagement_stable
THEN
  suggest(alternative_modules_same_level)
```

[9] Engagement and burnout rules

Burnout signal detection

```
IF session_duration >> norm
AND engagement ↓
AND performance ↓
THEN
  reduce_difficulty()
  suggest_short_modules()
  set_mode("RECOVERY")
```

Passive learning detection

```
IF low_interaction
AND repeated_fast_completion
THEN
  recommend(active_learning_format)
```

[10] Recommendation acceptance rules

Recommendation ignored pattern

```
IF recommendation_ignored_count >= K
THEN
  reduce_weight(recommendation_type)
```

Preferred path reinforcement

```
IF learner_consistently_chooses(path_type)
THEN
  increase_priority(path_type)
```

[11] Rule conflict resolution (global)

Priority enforcement

```
EVALUATE rules in priority order (1 → 4)

IF rule_triggered_at_priority(p)
THEN
  suppress_rules(priority > p)
```

Single decision mode

```
AT EACH DECISION:
  enforce(single_mode_only)
```

[12] Explainability rules

Decision logging

```
LOG {
  active_rules,
  suppressed_rules,
  final_decision,
  reason
}
```