SKILL-utf8

1 CC2.0 LEARN Function Skill

Function: LEARN - Continuous Learning & Knowledge Acquisition **Category Theory**: Profunctor + Contravariant Functor **Purpose**: Learn from outcomes, update knowledge, improve performance over time **Status**: =6 Planned for Phase 3 (Weeks 7-9)

1.1 When to Use This Skill

Use LEARN when you need to: - =Ú **Knowledge acquisition** - Learn from successes and failures - = **Continuous improvement** - Update strategies based on outcomes - =Ê **Pattern recognition** - Identify recurring patterns in data - <¯ **Model refinement** - Improve predictions and recommendations - >à **Transfer learning** - Apply knowledge across domains

---

1.2 Core Capabilities (Planned)

1.2.1 1. Outcome Learning (Profunctor)

```
// Learn from deployment outcomes
const learning = await learn({
  input: deploymentPlan,
  output: deploymentResult,
  outcome: { success: true, metrics: {...} }
});

// Updates internal knowledge base
// Returns: { patterns: [...], recommendations: [...] }
```

1.2.2 2. Pattern Recognition

```
// Identify patterns in historical data
const patterns = await learn.recognizePatterns({
  data: executionHistory,
  minSupport: 0.60,
  minConfidence: 0.80
});

// Returns: [
//   { pattern: "High quality   Low rollback rate", confidence: 0.92 },
//   { pattern: "Parallel execution   Token savings", confidence: 0.87 }
// ]
```

1.2.3 3. Knowledge Transfer

```
// Apply learning from one domain to another
const transferred = await learn.transfer({
  source: { domain: 'frontend', knowledge: reactPatterns },
  target: { domain: 'backend', context: nodeContext }
});
```

1.3 Planned Features

1.3.1 Learning Strategies

1. Supervised Learning - Learn from labeled examples (success/failure) - Improve predictions over time - Update confidence scores

2. Reinforcement Learning - Learn from rewards/penalties - Optimize decision-making policies - Explore vs exploit tradeoff

3. Transfer Learning - Apply knowledge across domains - Reduce learning time for new tasks - Leverage existing patterns

1.3.2 Knowledge Base

Templates: - Successful code patterns - Effective planning strategies - Quality improvement techniques - Deployment best practices

Anti-Patterns: - Known failure modes - Ineffective approaches - Common mistakes - Risk indicators

Correlations: - Quality metrics Deployment success - Planning depth Execution time - Team size Coordination overhead - Complexity Bug rate

---

1.4 Integration with Meta-Infrastructure

1.4.1 Entity-Based Learning

```
// Track learning curves per entity
await storage.developers.update(devId, {
  learningMetrics: {
    improvementRate: 0.12,  // 12% quality improvement over time
    patternMastery: {
      'refactoring': 0.87,
      'optimization': 0.72
    },
    transferEfficiency: 0.65
  }
});
```

1.4.2 Template Optimization Integration

```
// Learn which templates work best
const optimization = await templateOptimizer.learn({
  template: 'service-class',
  outcomes: usageHistory,
  context: projectContext
});

// Updates template based on successes/failures
```

1.5 Categorical Structure

1.5.1 Profunctor + Contravariant Functor

Profunctor Laws:

```
-- LEARN maps from inputs and outputs to knowledge
LEARN : (Input, Output)   Knowledge

-- Profunctor structure allows learning from both directions
dimap : (a'   a)   (b   b')   LEARN(a, b)   LEARN(a', b')
```

Contravariant Functor (for input transformation):

```
-- Transform inputs while learning
contramap : (a'   a)   LEARN(a)   LEARN(a')
```

Status: ó Will be verified in Phase 3 implementation

1.6 Example Workflows

1.6.1 Workflow 1: Template Improvement

```
// Learn from template usage outcomes
const improvement = await learn({
  template: 'api-controller',

  successes: [
    { usage: {...}, quality: 0.92 },
    { usage: {...}, quality: 0.89 }
  ],

  failures: [
    { usage: {...}, quality: 0.58, issues: [...] }
  ],

  analysis: 'statistical'
});

// Result: Updated template with improvements
// { optimizations: [...], expectedImprovement: 0.08 }
```

1.6.2 Workflow 2: Strategy Learning

```
// Learn optimal strategies from execution history
const strategyLearning = await learn({
  decisions: planningHistory,

  outcomes: executionResults,

  features: [
    'task_complexity',
    'team_size',
    'deadline_pressure',
    'risk_tolerance'
  ],

  model: 'decision-tree'
});

// Returns: Improved strategy selection model
```

1.6.3 Workflow 3: Cross-Domain Transfer

```
// Transfer frontend patterns to backend
const transfer = await learn.transfer({
  source: {
    domain: 'react-frontend',
    patterns: [
      'component-composition',
      'state-management',
      'error-boundaries'
    ]
  },

  target: {
    domain: 'express-backend',
    context: apiDesignContext
  }
});

// Result: Backend patterns inspired by frontend learnings
// { patterns: ['middleware-composition', 'state-controllers', 'error-
        middleware'] }
```

1.7 Learning Metrics

Improvement Rate: - Track quality improvement over time - Measure learning velocity - Identify plateau points

Pattern Mastery: - Per-pattern success rates - Confidence in pattern application - Transfer effectiveness

Knowledge Retention: - How long patterns remain effective - Decay rates over time - Refresh requirements

---

1.8 Performance Targets

- **Learning Speed**: Detect patterns within 20-50 examples
- **Accuracy**: >80% prediction accuracy for learned patterns
- **Transfer Efficiency**: >60% knowledge transfer success rate
- **Memory Efficiency**: <1GB for knowledge base
- **Update Latency**: <1s for incremental learning

1.9 Implementation Timeline

Phase 3 (Weeks 7-9): - Week 7: Core learning engine, pattern recognition - Week 8: Transfer learning, knowledge base - Week 9: Integration with all functions, documentation

Dependencies: - OBSERVE function (provides data for learning) - REASON function (applies learned knowledge) - CREATE function (uses learned templates) - ó VERIFY function (validates learning outcomes) - ó DEPLOY function (provides deployment outcomes)

1.10 Related Skills

- **cc2-observe**: Provides data for learning
- **cc2-reason**: Applies learned strategies
- **cc2-create**: Uses learned templates
- **cc2-verify**: Validates learning effectiveness
- **cc2-meta-orchestrator**: Manages LEARN in feedback loops

1.11 References

- Specification: `~/cc2.0/functions/learn/FUNCTION.md` (planned)
- Integration: `~/cc2.0/functions/learn/cc_integration.md` (planned)
- Phase 3 Roadmap: `~/cc2.0/PHASE-1-FOUNDATION-SPEC.md`
- Template Optimizer: `~/cc2.0/implementations/agents/template-optimizer.ts` (existing)

1.12 Learning Cycle

```
// Complete feedback loop with LEARN
async function continuousImprovement(task: Task): Promise<Outcome> {
  // 1. OBSERVE current state
  const observation = await observe(task);

  // 2. REASON with learned knowledge
  const plan = await reason(observation, {
    knowledgeBase: await learn.getKnowledge()
  });

  // 3. CREATE using learned templates
  const artifact = await create(plan, {
    templates: await learn.getBestTemplates(plan.strategy)
  });

  // 4. VERIFY quality
  const verification = await verify(artifact);

  // 5. DEPLOY to production
  const deployment = await deploy(artifact);

  // 6. LEARN from outcome
  await learn({
    input: { observation, plan, artifact },
    output: { verification, deployment },
    outcome: deployment.metrics
  });

  // Knowledge base now updated for next iteration
  return deployment;
}
```

1.13 Meta-Function: LEARN_SELF

LEARN will eventually observe and improve its own learning patterns:

```
// Meta-learning (planned)
const metaLearn = await learnSelf(learningHistory);

// Returns insights about learning effectiveness
{
  learningRate: 0.08,  // 8% improvement per 100 examples
  transferSuccess: 0.62,
  bestStrategy: "supervised-with-reinforcement",
  suggestions: [
    "Increase exploration rate for novel patterns",
    "Add cross-validation for pattern recognition",
    "Implement curriculum learning for complex domains"
  ]
}
```

This enables **meta-learning**: learning how to learn better.

1.14 Notes

This skill represents planned functionality for Phase 3 of CC2.0 development. The categorical structure (Profunctor + Contravariant Functor) and learning patterns are designed, but implementation is pending completion of all other functions to provide rich learning data.

LEARN is the final piece that closes the feedback loop, enabling true continuous improvement:

```
OBSERVE   REASON   CREATE   VERIFY   DEPLOY   LEARN   [loop back to
OBSERVE]
```

Expected Delivery: Week 9 of Phase 1 implementation plan

1.15 Vision

With LEARN complete, CC2.0 becomes a **self-improving system**:

- Learns from every execution
- Improves strategies over time
- Transfers knowledge across domains
- Optimizes its own performance
- Achieves mastery through continuous practice

This is the essence of artificial intelligence: not just executing tasks, but **learning to execute them better**.