Lottery Prize Tier Targeting System - Detailed Output Specification

Document Overview

Document Type: Technical Output Specification
Version: 2.0
Date: December 2024
Purpose: Complete specification for implementing lottery number generation with pattern string grouping and intelligent stopping criteria

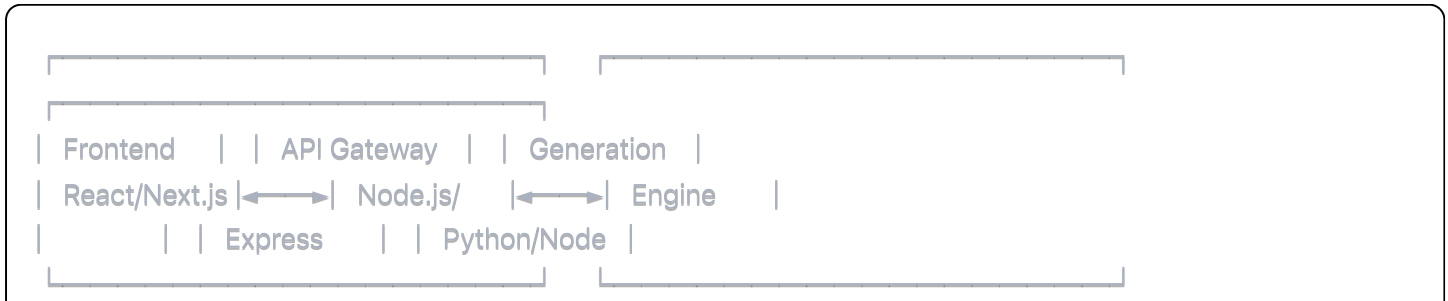
---

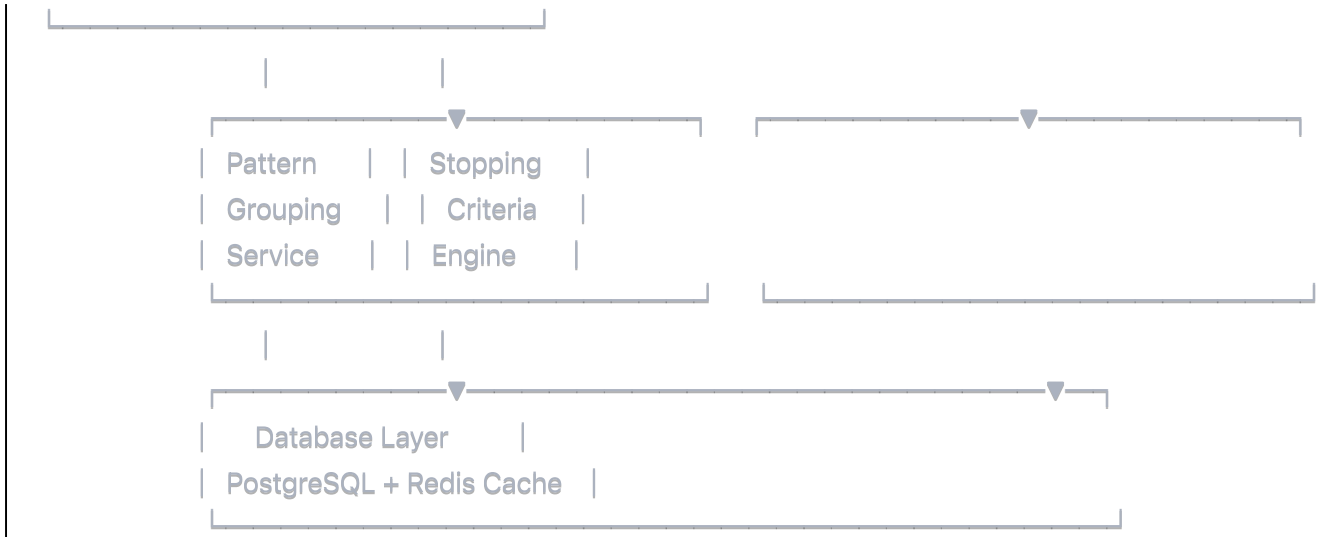
Table of Contents

---

1. System Architecture Overview {#system-architecture}

1.1 High-Level Architecture





1.2 Core Components

- **Generation Engine**: Core number generation with pattern integration
- **Pattern Grouping Service**: Hot/Warm/Cold pattern classification
- **Stopping Criteria Engine**: Intelligent generation termination logic
- **Configuration Service**: Dynamic threshold and strategy management
- **Analytics Engine**: Real-time quality metrics and performance tracking

---

2. Data Models & Structures {#data-models}

2.1 Core Data Types

LotteryConfiguration

typescript

```typescript
interface LotteryConfiguration {
  id: string;
  name: string;
  numberRange: {
    min: number;       // 1 for 1-35 lottery
    max: number;       // 35 for 1-35 lottery
  };
  drawSize: number;    // 5 for pick-5 lottery
  patternLength: number; // 5 for 5-digit patterns
  prizeStructure: {
    tier3: { matches: 3; prize: number; };
    tier4: { matches: 4; prize: number; };
    tier5: { matches: 5; prize: number; };
  };
}
```

PatternGroupDefinition

typescript

```typescript
interface PatternGroupDefinition {
  lotteryConfigId: string;
  groups: {
    hot: {
      patterns: string[];    // ['01123', '01223', '01122', ...]
      frequency: number[];   // Historical frequency for each pattern
      efficiency: number;    // 2.11 for hot patterns
      lastUpdated: Date;
    };
    warm: {
      patterns: string[];    // ['11223', '01233', '01222', ...]
      frequency: number[];
      efficiency: number;    // 1.5 for warm patterns
      lastUpdated: Date;
    };
    cold: {
      patterns: string[];    // ['12223', '02233', '01133', ...]
      frequency: number[];
      efficiency: number;    // 0.8 for cold patterns
      lastUpdated: Date;
    };
  };
  totalAnalyzedDraws: number;
  lastAnalysisDate: Date;
}
```

UserRequest

typescript

```typescript
interface UserRequest {
  sessionId: string;
  targetTier: 'tier3' | 'tier4' | 'tier5';
  numberOfTickets: number;       // 1-500
  strategy: 'conservative' | 'balanced' | 'aggressive';
  budget: number;                // Total willing to spend
  lotteryConfig: string;         // Reference to LotteryConfiguration.id
  preferences?: {
    maxGenerationTime?: number;  // Override default timeout
    qualityThreshold?: number;   // Override minimum quality
    allowColdPatterns?: boolean; // Allow cold pattern usage
  };
  timestamp: Date;
}
```

GeneratedTicketSet

typescript

```typescript
interface GeneratedTicketSet {
  sessionId: string;
  tickets: LotteryTicket[];
  generation: GenerationMetadata;
  quality: QualityMetrics;
  patterns: PatternAnalysis;
  expectedPerformance: PerformanceProjection;
  status: 'success' | 'partial' | 'fallback' | 'failed';
  timestamp: Date;
}

interface LotteryTicket {
  id: string;
  numbers: number[];             // [1, 15, 23, 28, 34]
  pattern: string;               // "01223"
  patternGroup: 'hot' | 'warm' | 'cold' | 'unknown';
  generationStrategy: string;    // "hot_pair", "triplet_foundation", etc.
  qualityScore: number;          // 0.0 - 1.0
  expectedMatches: number;       // Predicted matches for this ticket
}
```

QualityMetrics

typescript

```typescript
interface QualityMetrics {
  overall: {
    optimizationScore: number;    // 0.0 - 1.0
    qualityGrade: 'A+' | 'A' | 'B+' | 'B' | 'C+' | 'C' | 'D';
    passedAllThresholds: boolean;
  };
  patternGrouping: {
    hotPatternCoverage: number;        // % of tickets using hot patterns
    warmPatternCoverage: number;       // % of tickets using warm patterns
    coldPatternCoverage: number;       // % of tickets using cold patterns
    unknownPatternCoverage: number;    // % of tickets using unknown patterns
    patternDiversity: {
      uniqueHotPatterns: number;       // Number of different hot patterns used
      uniqueWarmPatterns: number;      // Number of different warm patterns used
      maxSinglePatternUsage: number;   // Highest usage count for any pattern
      diversityScore: number;          // 0.0 - 1.0
    };
    patternEfficiency: number;         // Weighted efficiency based on pattern mix
  };
  numberAnalysis: {
    hotNumberCoverage: number;         // % of tickets with hot numbers
    numberDistribution: {
      range1_7: number;                // Count in range 1-7
      range8_14: number;               // Count in range 8-14
      range15_21: number;              // Count in range 15-21
      range22_28: number;              // Count in range 22-28
      range29_35: number;              // Count in range 29-35
    };
    consecutiveNumberRate: number;     // % of tickets with consecutive numbers
    provenCombinationUsage: {
      frequentPairs: number;           // Count of frequent pair usage
      provenTriplets: number;          // Count of proven triplet usage
      overlappingPatterns: number;     // Count of overlapping pattern usage
    };
  };
  diversity: {
    uniqueNumbers: number;             // Total unique numbers across all tickets
    averageOverlap: number;            // Average number overlap between tickets
    coverageRatio: number;             // uniqueNumbers / totalPossibleNumbers
  };
}
```

GenerationMetadata

typescript

```typescript
interface GenerationMetadata {
  attempts: number;                // Number of generation attempts
  timeElapsed: number;             // Generation time in milliseconds
  stoppingReason: 'pattern_thresholds' | 'quality_thresholds' | 'max_attempts' |
                  'timeout' | 'quality_stall' | 'user_override';
  algorithmVersion: string;        // "2.1.0"
  strategyDistribution: {
    hot_pair: number;              // Number of tickets using each strategy
    triplet_foundation: number;
    consecutive_enhanced: number;
    pattern_specific: number;
    balanced_range: number;
  };
  qualityEvolution: {
    attempt: number;
    qualityScore: number;
    patternCoverage: number;
    timestamp: number;
  }[];
}
```

StoppingCriteria

typescript

```
interface StoppingCriteria {
  patternGrouping: {
    hotPatternCoverage: {
      minimum: number;              // 0.70 for tier3, 0.85 for tier4, 0.95 for tier5
      weight: number;               // Importance weight (1.0 = critical)
    };
    warmPatternLimit: {
      maximum: number;              // 0.20 for tier3, 0.12 for tier4, 0.05 for tier5
      weight: number;
    };
    coldPatternAllowance: {
      maximum: number;              // 0.10 for tier3, 0.03 for tier4, 0.00 for tier5
      weight: number;
    };
    patternDiversity: {
      minimumUniqueHot: number;     // Minimum different hot patterns required
      maximumSinglePattern: number; // Maximum tickets for any single pattern
      weight: number;
    };
    patternEfficiency: {
      minimum: number;              // Minimum weighted efficiency score
      weight: number;
    };
  };
  traditional: {
    optimizationScore: {
      minimum: number;              // 0.65 for tier3, 0.70 for tier4, 0.75 for tier5
      weight: number;
    };
    hotNumberCoverage: {
      minimum: number;              // Strategy-adjusted hot number usage
      weight: number;
    };
    diversityBalance: {
      minimum: number;              // Minimum diversity for conservative
      maximum: number;              // Maximum concentration for aggressive
      weight: number;
    };
  };
  meta: {
    maxAttempts: number;            // Maximum generation attempts
    timeoutMs: number;              // Maximum generation time
    stallThreshold: number;         // Attempts without improvement
    minimumQualityGate: number;     // Never stop below this quality (0.60)
```

```
  };
}
```

---

3. API Specifications {#api-specifications}

3.1 Core Generation API

POST /api/v2/generate-tickets

Request Body:

```json
{
  "targetTier": "tier4",
  "numberOfTickets": 100,
  "strategy": "aggressive",
  "budget": 500,
  "lotteryType": "5/35",
  "preferences": {
    "maxGenerationTime": 30000,
    "qualityThreshold": 0.75
  }
}
```

Response (200 OK):

```json
```

```
{
  "sessionId": "uuid-v4",
  "status": "success",
  "tickets": [
    {
      "id": "ticket-uuid-1",
      "numbers": [5, 16, 19, 23, 28],
      "pattern": "01122",
      "patternGroup": "hot",
      "generationStrategy": "hot_pair",
      "qualityScore": 0.847,
      "expectedMatches": 3.2
    }
  ],
  "generation": {
    "attempts": 23,
    "timeElapsed": 12300,
    "stoppingReason": "pattern_thresholds",
    "algorithmVersion": "2.1.0",
    "strategyDistribution": {
      "hot_pair": 60,
      "triplet_foundation": 40,
      "consecutive_enhanced": 0,
      "pattern_specific": 0,
      "balanced_range": 0
    }
  },
  "quality": {
    "overall": {
      "optimizationScore": 0.823,
      "qualityGrade": "A",
      "passedAllThresholds": true
    },
    "patternGrouping": {
      "hotPatternCoverage": 0.94,
      "warmPatternCoverage": 0.06,
      "coldPatternCoverage": 0.00,
      "unknownPatternCoverage": 0.00,
      "patternDiversity": {
        "uniqueHotPatterns": 4,
        "uniqueWarmPatterns": 1,
        "maxSinglePatternUsage": 31,
        "diversityScore": 0.67
      },
      "patternEfficiency": 0.89
    }
```

```
  },
  "expectedPerformance": {
    "theoretical": {
      "tier3Wins": 0.65,
      "tier4Wins": 0.065,
      "tier5Wins": 0.0003
    },
    "optimized": {
      "tier3Wins": 0.78,
      "tier4Wins": 0.078,
      "tier5Wins": 0.00036
    },
    "improvement": {
      "tier3": "20.0%",
      "tier4": "20.0%",
      "tier5": "20.0%"
    },
    "financialProjection": {
      "expectedRevenue": 195.50,
      "breakEvenProbability": 0.003,
      "roi": -80.45
    }
  }
}
```


Error Responses:

json

```
// 400 Bad Request
{
  "error": "INVALID_REQUEST",
  "message": "Number of tickets must be between 1 and 500",
  "code": 4001,
  "details": {
    "field": "numberOfTickets",
    "provided": 1000,
    "allowed": { "min": 1, "max": 500 }
  }
}

// 422 Generation Failed
{
  "error": "GENERATION_FAILED",
  "message": "Unable to meet pattern thresholds within time limit",
  "code": 4221,
  "details": {
    "attempts": 300,
    "timeElapsed": 30000,
    "bestQuality": 0.67,
    "requiredQuality": 0.75,
    "suggestions": [
      "Reduce number of tickets",
      "Lower strategy from aggressive to balanced",
      "Increase time limit"
    ]
  }
}
```

3.2 Pattern Analysis API

GET /api/v2/pattern-analysis/{lotteryType}

Response:

```
json
```

```
{
  "lotteryType": "5/35",
  "lastUpdated": "2024-12-01T10:30:00Z",
  "totalAnalyzedDraws": 5000,
  "patternGroups": {
    "hot": {
      "patterns": [
        {
          "pattern": "01123",
          "frequency": 387,
          "percentage": 7.74,
          "efficiency": 2.15,
          "lastSeen": 13,
          "averageGap": 12.9
        }
      ],
      "totalPatterns": 6,
      "combinedFrequency": 2234,
      "averageEfficiency": 2.11
    }
  },
  "hotNumbers": [
    {
      "number": 20,
      "frequency": 120,
      "percentage": 17.8,
      "patternAssociation": ["01223", "11223", "12223"]
    }
  ],
  "provenCombinations": {
    "frequentPairs": [
      {
        "pair": [19, 28],
        "frequency": 24,
        "percentage": 3.6,
        "associatedPatterns": ["01223", "12223"]
      }
    ],
    "provenTriplets": [
      {
        "triplet": [8, 13, 25],
        "frequency": 6,
        "percentage": 0.9,
        "associatedPatterns": ["01222", "00123"]
      }
    ]
```

```
  }
}
```

3.3 Real-time Generation Status API

GET /api/v2/generation-status/{sessionId}

Response (Server-Sent Events):

```javascript
// Event: progress
data: {
  "sessionId": "uuid",
  "attempt": 15,
  "currentQuality": 0.67,
  "patternCoverage": {
    "hot": 0.83,
    "warm": 0.12,
    "cold": 0.05
  },
  "timeElapsed": 8500,
  "estimatedTimeRemaining": 12000,
  "status": "generating"
}

// Event: threshold_met
data: {
  "sessionId": "uuid",
  "thresholdType": "pattern_grouping",
  "thresholdName": "hotPatternCoverage",
  "achieved": 0.94,
  "required": 0.85,
  "timestamp": "2024-12-01T10:35:23Z"
}

// Event: complete
data: {
  "sessionId": "uuid",
  "status": "success",
  "finalQuality": 0.823,
  "totalAttempts": 23,
  "totalTime": 12300
}
```

4. Core Algorithm Specifications {#algorithms}

4.1 Master Generation Algorithm

```python
```

```
class MasterGenerationEngine:
    def generate_tickets(self, request: UserRequest) -> GeneratedTicketSet:
        # 1. Initialize
        session = self.initialize_session(request)
        criteria = self.calculate_stopping_criteria(request)
        pattern_groups = self.load_pattern_groups(request.lotteryConfig)

        # 2. Generation Loop
        attempts = 0
        best_result = None
        stall_counter = 0

        while attempts < criteria.meta.maxAttempts:
            attempts += 1

            # Generate ticket batch
            tickets = self.generate_ticket_batch(
                request, pattern_groups, attempts
            )

            # Evaluate quality
            quality = self.evaluate_quality(tickets, criteria, request)

            # Check stopping conditions (PRIMARY: Pattern Grouping)
            if self.check_pattern_grouping_thresholds(quality, criteria):
                return self.create_success_result(
                    tickets, quality, attempts, "pattern_thresholds"
                )

            # Check traditional thresholds (SECONDARY)
            if self.check_traditional_thresholds(quality, criteria):
                return self.create_success_result(
                    tickets, quality, attempts, "quality_thresholds"
                )

            # Check meta conditions (FALLBACK)
            if self.check_meta_conditions(attempts, quality, criteria):
                return self.create_fallback_result(
                    tickets, quality, attempts
                )

            # Track best result and stalling
            if self.is_better_quality(quality, best_result):
                best_result = (tickets, quality)
                stall_counter = 0
            else:
```

```
            stall_counter += 1

        # Return best available result
        return self.create_fallback_result(
            best_result[0], best_result[1], attempts
        )
```

4.2 Pattern Grouping Evaluation

python

```python
def check_pattern_grouping_thresholds(
    self, quality: QualityMetrics, criteria: StoppingCriteria
) -> bool:
    tests = [
        # Hot pattern coverage (CRITICAL)
        quality.patternGrouping.hotPatternCoverage >=
        criteria.patternGrouping.hotPatternCoverage.minimum,

        # Warm pattern limit
        quality.patternGrouping.warmPatternCoverage <=
        criteria.patternGrouping.warmPatternLimit.maximum,

        # Cold pattern allowance
        quality.patternGrouping.coldPatternCoverage <=
        criteria.patternGrouping.coldPatternAllowance.maximum,

        # Pattern diversity
        quality.patternGrouping.patternDiversity.uniqueHotPatterns >=
        criteria.patternGrouping.patternDiversity.minimumUniqueHot,

        # Pattern concentration limit
        quality.patternGrouping.patternDiversity.maxSinglePatternUsage <=
        criteria.patternGrouping.patternDiversity.maximumSinglePattern,

        # Pattern efficiency
        quality.patternGrouping.patternEfficiency >=
        criteria.patternGrouping.patternEfficiency.minimum
    ]

    # All pattern grouping tests must pass
    return all(tests)
```

4.3 Ticket Generation Strategies

python

```python
class TicketGenerationStrategies:
    def generate_ticket_batch(
        self, request: UserRequest, pattern_groups: PatternGroupDefinition,
        attempt: int
    ) -> List[LotteryTicket]:

        strategy_distribution = self.calculate_strategy_distribution(
            request.targetTier, request.strategy, request.numberOfTickets
        )

        tickets = []

        # Hot pattern foundation (60% of tickets for tier4 aggressive)
        hot_tickets = self.generate_hot_pattern_tickets(
            strategy_distribution['hot_foundation'],
            pattern_groups.groups.hot,
            request
        )
        tickets.extend(hot_tickets)

        # Triplet enhancement (25% of tickets)
        triplet_tickets = self.generate_triplet_tickets(
            strategy_distribution['triplet_enhancement'],
            pattern_groups,
            request
        )
        tickets.extend(triplet_tickets)

        # Pair optimization (15% of tickets)
        pair_tickets = self.generate_pair_tickets(
            strategy_distribution['pair_optimization'],
            pattern_groups,
            request
        )
        tickets.extend(pair_tickets)

        return self.ensure_uniqueness(tickets, request.numberOfTickets)

    def generate_hot_pattern_tickets(
        self, count: int, hot_patterns: dict, request: UserRequest
    ) -> List[LotteryTicket]:
        tickets = []

        for i in range(count):
            # Select pattern based on frequency weighting
            pattern = self.select_weighted_pattern(hot_patterns.patterns,
```

```
                    hot_patterns.frequency)

        # Generate numbers for this pattern
        numbers = self.generate_numbers_for_pattern(pattern, request)

        # Create ticket
        ticket = LotteryTicket(
            id=f"hot-{i}",
            numbers=numbers,
            pattern=pattern,
            patternGroup="hot",
            generationStrategy="hot_pattern_foundation",
            qualityScore=self.calculate_ticket_quality(numbers, pattern),
            expectedMatches=self.calculate_expected_matches(numbers, pattern)
        )
        tickets.append(ticket)

    return tickets
```

4.4 Pattern-to-Numbers Generation

python

```python
def generate_numbers_for_pattern(
    self, pattern: str, request: UserRequest
) -> List[int]:
    numbers = []
    used_numbers = set()

    for position, digit_char in enumerate(pattern):
        digit = int(digit_char)

        # Define number ranges for each digit
        if digit == 0:   # 1-9 for single digits
            range_start, range_end = 1, 9
        elif digit == 1: # 10-19
            range_start, range_end = 10, 19
        elif digit == 2: # 20-29
            range_start, range_end = 20, 29
        elif digit == 3: # 30-35 (for 5/35 lottery)
            range_start, range_end = 30, 35
        else:
            continue  # Skip invalid digits

        # Get hot numbers in this range
        hot_numbers_in_range = [
            n for n in self.hot_numbers
            if range_start <= n <= range_end and n not in used_numbers
        ]

        # Selection strategy based on user preference
        if request.strategy == 'aggressive' and hot_numbers_in_range:
            # 90% chance to use hot number
            if random.random() < 0.9:
                selected = random.choice(hot_numbers_in_range)
            else:
                selected = self.select_random_from_range(
                    range_start, range_end, used_numbers
                )
        elif request.strategy == 'balanced' and hot_numbers_in_range:
            # 70% chance to use hot number
            if random.random() < 0.7:
                selected = random.choice(hot_numbers_in_range)
            else:
                selected = self.select_random_from_range(
                    range_start, range_end, used_numbers
                )
        else:
            # Conservative: 50% chance or no hot numbers available
```

```
            if hot_numbers_in_range and random.random() < 0.5:
                selected = random.choice(hot_numbers_in_range)
            else:
                selected = self.select_random_from_range(
                    range_start, range_end, used_numbers
                )

            numbers.append(selected)
            used_numbers.add(selected)

        return sorted(numbers)
```

---

5. Pattern Grouping Engine {#pattern-engine}

5.1 Pattern Classification Service

python

```python
class PatternGroupingEngine:
    def __init__(self, lottery_config: LotteryConfiguration):
        self.config = lottery_config
        self.pattern_groups = self.load_pattern_groups()
        self.hot_numbers = self.load_hot_numbers()

    def classify_ticket_pattern(self, numbers: List[int]) -> dict:
        pattern = self.extract_pattern(numbers)

        classification = {
            'pattern': pattern,
            'group': self.determine_pattern_group(pattern),
            'frequency': self.get_pattern_frequency(pattern),
            'efficiency': self.get_pattern_efficiency(pattern),
            'numbers_analysis': {
                'hot_count': self.count_hot_numbers(numbers),
                'consecutive_pairs': self.count_consecutive_pairs(numbers),
                'range_distribution': self.analyze_range_distribution(numbers)
            }
        }

        return classification

    def extract_pattern(self, numbers: List[int]) -> str:
        """Convert [5, 16, 23, 28, 34] to '01223'"""
        return ''.join([str(num // 10) for num in sorted(numbers)])

    def determine_pattern_group(self, pattern: str) -> str:
        if pattern in self.pattern_groups['hot']['patterns']:
            return 'hot'
        elif pattern in self.pattern_groups['warm']['patterns']:
            return 'warm'
        elif pattern in self.pattern_groups['cold']['patterns']:
            return 'cold'
        else:
            return 'unknown'

    def analyze_ticket_set_patterns(
        self, tickets: List[LotteryTicket]
    ) -> PatternAnalysis:

        group_counts = {'hot': 0, 'warm': 0, 'cold': 0, 'unknown': 0}
        pattern_usage = {}
        unique_patterns_by_group = {'hot': set(), 'warm': set(), 'cold': set()}

        for ticket in tickets:
```

```
        # Count by group
        group_counts[ticket.patternGroup] += 1

        # Track pattern usage
        if ticket.pattern not in pattern_usage:
            pattern_usage[ticket.pattern] = 0
        pattern_usage[ticket.pattern] += 1

        # Track unique patterns by group
        if ticket.patternGroup in unique_patterns_by_group:
            unique_patterns_by_group[ticket.patternGroup].add(ticket.pattern)

    total_tickets = len(tickets)
    max_single_usage = max(pattern_usage.values()) if pattern_usage else 0

    return PatternAnalysis(
        hotPatternCoverage=group_counts['hot'] / total_tickets,
        warmPatternCoverage=group_counts['warm'] / total_tickets,
        coldPatternCoverage=group_counts['cold'] / total_tickets,
        unknownPatternCoverage=group_counts['unknown'] / total_tickets,
        patternDiversity=PatternDiversity(
            uniqueHotPatterns=len(unique_patterns_by_group['hot']),
            uniqueWarmPatterns=len(unique_patterns_by_group['warm']),
            maxSinglePatternUsage=max_single_usage,
            diversityScore=len(pattern_usage) / total_tickets
        ),
        patternEfficiency=self.calculate_weighted_efficiency(tickets)
    )
```

5.2 Pattern Database Management

sql

```sql
-- Pattern Groups Table
CREATE TABLE pattern_groups (
    id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    lottery_config_id UUID NOT NULL REFERENCES lottery_configurations(id),
    pattern VARCHAR(10) NOT NULL,
    group_type VARCHAR(10) NOT NULL CHECK (group_type IN ('hot', 'warm', 'cold')),
    frequency INTEGER NOT NULL DEFAULT 0,
    historical_percentage DECIMAL(5,3) NOT NULL DEFAULT 0.0,
    efficiency_multiplier DECIMAL(4,2) NOT NULL DEFAULT 1.0,
    last_seen_draw INTEGER,
    average_gap DECIMAL(6,2),
    created_at TIMESTAMP DEFAULT NOW(),
    updated_at TIMESTAMP DEFAULT NOW(),
    UNIQUE(lottery_config_id, pattern)
);

-- Hot Numbers Table
CREATE TABLE hot_numbers (
    id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    lottery_config_id UUID NOT NULL REFERENCES lottery_configurations(id),
    number INTEGER NOT NULL,
    frequency INTEGER NOT NULL DEFAULT 0,
    historical_percentage DECIMAL(5,3) NOT NULL DEFAULT 0.0,
    associated_patterns TEXT[] NOT NULL DEFAULT '{}',
    last_seen_draw INTEGER,
    average_gap DECIMAL(6,2),
    created_at TIMESTAMP DEFAULT NOW(),
    updated_at TIMESTAMP DEFAULT NOW(),
    UNIQUE(lottery_config_id, number)
);

-- Proven Combinations Table
CREATE TABLE proven_combinations (
    id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    lottery_config_id UUID NOT NULL REFERENCES lottery_configurations(id),
    combination_type VARCHAR(20) NOT NULL CHECK (combination_type IN ('pair', 'triplet')),
    numbers INTEGER[] NOT NULL,
    frequency INTEGER NOT NULL DEFAULT 0,
    historical_percentage DECIMAL(5,3) NOT NULL DEFAULT 0.0,
    associated_patterns TEXT[] NOT NULL DEFAULT '{}',
    last_seen_draw INTEGER,
    created_at TIMESTAMP DEFAULT NOW(),
    updated_at TIMESTAMP DEFAULT NOW()
);
```

6. Stopping Criteria Engine {#stopping-engine}

6.1 Criteria Calculation Engine

python

```python
class StoppingCriteriaEngine:
    def calculate_criteria(
        self, request: UserRequest
    ) -> StoppingCriteria:

        base_criteria = self.get_base_criteria(request.targetTier)
        strategy_modifiers = self.get_strategy_modifiers(request.strategy)
        volume_adjustments = self.calculate_volume_adjustments(
            request.numberOfTickets, request.budget
        )

        # Apply modifications in order
        adjusted_criteria = self.apply_strategy_modifications(
            base_criteria, strategy_modifiers
        )
        final_criteria = self.apply_volume_adjustments(
            adjusted_criteria, volume_adjustments
        )

        # Add meta criteria
        final_criteria.meta = self.calculate_meta_criteria(
            request.numberOfTickets, request.strategy
        )

        return final_criteria

    def get_base_criteria(self, target_tier: str) -> dict:
        base_criteria = {
            'tier3': {
                'patternGrouping': {
                    'hotPatternCoverage': {'minimum': 0.70, 'weight': 1.0},
                    'warmPatternLimit': {'maximum': 0.20, 'weight': 0.8},
                    'coldPatternAllowance': {'maximum': 0.10, 'weight': 0.6},
                    'patternDiversity': {
                        'minimumUniqueHot': 4,  # 4 of 6 hot patterns
                        'maximumSinglePattern': 0.40,  # 40% max concentration
                        'weight': 0.7
                    },
                    'patternEfficiency': {'minimum': 0.65, 'weight': 0.9}
                }
            },
            'tier4': {
                'patternGrouping': {
                    'hotPatternCoverage': {'minimum': 0.85, 'weight': 1.0},
                    'warmPatternLimit': {'maximum': 0.12, 'weight': 0.8},
                    'coldPatternAllowance': {'maximum': 0.03, 'weight': 0.9},
```

```
            'patternDiversity': {
                'minimumUniqueHot': 3,  # 3 of 6 hot patterns
                'maximumSinglePattern': 0.50,  # 50% max concentration
                'weight': 0.6
            },
            'patternEfficiency': {'minimum': 0.75, 'weight': 1.0}
        }
    },
    'tier5': {
        'patternGrouping': {
            'hotPatternCoverage': {'minimum': 0.95, 'weight': 1.0},
            'warmPatternLimit': {'maximum': 0.05, 'weight': 0.9},
            'coldPatternAllowance': {'maximum': 0.00, 'weight': 1.0},
            'patternDiversity': {
                'minimumUniqueHot': 2,  # 2 of 6 hot patterns (concentration)
                'maximumSinglePattern': 0.60,  # 60% max concentration
                'weight': 0.5
            },
            'patternEfficiency': {'minimum': 0.85, 'weight': 1.0}
        }
    }
}

return base_criteria[target_tier]
```

6.2 Real-time Evaluation Engine

python

```python
class QualityEvaluationEngine:
    def evaluate_quality(
        self, tickets: List[LotteryTicket],
        criteria: StoppingCriteria,
        request: UserRequest
    ) -> QualityMetrics:

        # Pattern grouping analysis (PRIMARY)
        pattern_analysis = self.pattern_engine.analyze_ticket_set_patterns(tickets)

        # Number analysis (SECONDARY)
        number_analysis = self.analyze_number_quality(tickets)

        # Diversity analysis (SECONDARY)
        diversity_analysis = self.analyze_diversity(tickets)

        # Calculate overall optimization score
        optimization_score = self.calculate_optimization_score(
            pattern_analysis, number_analysis, diversity_analysis, request.targetTier
        )

        return QualityMetrics(
            overall=OverallQuality(
                optimizationScore=optimization_score,
                qualityGrade=self.calculate_quality_grade(optimization_score),
                passedAllThresholds=self.check_all_thresholds(
                    pattern_analysis, criteria
                )
            ),
            patternGrouping=pattern_analysis,
            numberAnalysis=number_analysis,
            diversity=diversity_analysis
        )

    def calculate_optimization_score(
        self, pattern_analysis: PatternAnalysis,
        number_analysis: NumberAnalysis,
        diversity_analysis: DiversityAnalysis,
        target_tier: str
    ) -> float:

        # Weight distribution based on target tier
        weights = {
            'tier3': {'pattern': 0.5, 'number': 0.3, 'diversity': 0.2},
            'tier4': {'pattern': 0.7, 'number': 0.2, 'diversity': 0.1},
            'tier5': {'pattern': 0.8, 'number': 0.15, 'diversity': 0.05}
```

```
    }[target_tier]

    # Pattern score (most important)
    pattern_score = (
        pattern_analysis.hotPatternCoverage * 0.4 +
        pattern_analysis.patternEfficiency * 0.3 +
        pattern_analysis.patternDiversity.diversityScore * 0.2 +
        (1.0 - pattern_analysis.coldPatternCoverage) * 0.1
    )

    # Number score
    number_score = (
        number_analysis.hotNumberCoverage * 0.5 +
        number_analysis.provenCombinationUsage.frequentPairs * 0.3 +
        number_analysis.consecutiveNumberRate * 0.2
    )

    # Diversity score
    diversity_score = (
        diversity_analysis.coverageRatio * 0.6 +
        (1.0 - diversity_analysis.averageOverlap / 5.0) * 0.4
    )

    # Weighted combination
    final_score = (
        pattern_score * weights['pattern'] +
        number_score * weights['number'] +
        diversity_score * weights['diversity']
    )

    return min(1.0, max(0.0, final_score))
```

---

7. User Interface Specifications {#ui-specifications}

7.1 Input Interface Components

TierSelector Component

```
tsx
```

```tsx
interface TierSelectorProps {
  value: 'tier3' | 'tier4' | 'tier5';
  onChange: (tier: string) => void;
  showExpectedWins?: boolean;
}

const TierSelector: React.FC<TierSelectorProps> = ({
  value, onChange, showExpectedWins = true
}) => {
  const tiers = [
    {
      id: 'tier3',
      name: 'Tier 3 (3 matches)',
      description: 'Most frequent wins, $15 prize',
      expectedRate: '2.25%',
      strategy: 'Coverage optimization',
      color: 'green'
    },
    {
      id: 'tier4',
      name: 'Tier 4 (4 matches)',
      description: 'Balanced frequency, $500 prize',
      expectedRate: '0.065%',
      strategy: 'Pattern concentration',
      color: 'blue'
    },
    {
      id: 'tier5',
      name: 'Tier 5 (5 matches)',
      description: 'Rare jackpot, $100,000 prize',
      expectedRate: '0.0003%',
      strategy: 'Maximum overlap',
      color: 'purple'
    }
  ];

  return (
    <div className="tier-selector">
      {tiers.map(tier => (
        <TierCard
          key={tier.id}
          tier={tier}
          selected={value === tier.id}
          onClick={() => onChange(tier.id)}
          showExpectedWins={showExpectedWins}
        />
```

```
      ))}
    </div>
  );
};
```

StrategySelector Component

tsx

```tsx
interface StrategySelectorProps {
  value: 'conservative' | 'balanced' | 'aggressive';
  onChange: (strategy: string) => void;
  targetTier: string;
}

const StrategySelector: React.FC<StrategySelectorProps> = ({
  value, onChange, targetTier
}) => {
  const strategies = [
    {
      id: 'conservative',
      name: 'Conservative',
      description: 'Maximum coverage, lower risk',
      characteristics: {
        diversification: 'High',
        hotPatternUsage: '60%',
        riskLevel: 'Low'
      },
      bestFor: 'First-time players, risk-averse',
      icon: 'shield'
    },
    {
      id: 'balanced',
      name: 'Balanced',
      description: 'Optimal risk/reward ratio',
      characteristics: {
        diversification: 'Medium',
        hotPatternUsage: '75%',
        riskLevel: 'Medium'
      },
      bestFor: 'Most players, best overall value',
      icon: 'scales',
      recommended: true
    },
    {
      id: 'aggressive',
      name: 'Aggressive',
      description: 'Maximum pattern concentration',
      characteristics: {
        diversification: 'Low',
        hotPatternUsage: '90%',
        riskLevel: 'High'
      },
      bestFor: 'High-budget, thrill-seekers',
      icon: 'target'
```

```
  }
 ];

 return (
  <StrategyGrid
   strategies={strategies}
   selected={value}
   onSelect={onChange}
   targetTier={targetTier}
  />
 );
};
```

7.2 Generation Progress Interface

GenerationProgress Component

tsx

```tsx
interface GenerationProgressProps {
  sessionId: string;
  onComplete: (result: GeneratedTicketSet) => void;
  onError: (error: GenerationError) => void;
}

const GenerationProgress: React.FC<GenerationProgressProps> = ({
  sessionId, onComplete, onError
}) => {
  const [status, setStatus] = useState<GenerationStatus>();

  useEffect(() => {
    const eventSource = new EventSource(`/api/v2/generation-status/${sessionId}`);

    eventSource.onmessage = (event) => {
      const data = JSON.parse(event.data);

      switch (event.type) {
        case 'progress':
          setStatus(data);
          break;
        case 'threshold_met':
          // Show threshold achievement notification
          showThresholdNotification(data);
          break;
        case 'complete':
          onComplete(data);
          break;
        case 'error':
          onError(data);
          break;
      }
    };

    return () => eventSource.close();
  }, [sessionId]);

  return (
    <div className="generation-progress">
      <ProgressHeader
        status={status?.status}
        timeElapsed={status?.timeElapsed}
        estimatedRemaining={status?.estimatedTimeRemaining}
      />

      <QualityProgressBar
```

```
        current={status?.currentQuality}
        target={status?.targetQuality}
        label="Overall Quality"
      />

      <PatternProgressSection
        hotCoverage={status?.patternCoverage?.hot}
        warmCoverage={status?.patternCoverage?.warm}
        coldCoverage={status?.patternCoverage?.cold}
        requirements={status?.requirements}
      />

      <AttemptCounter
        current={status?.attempt}
        maximum={status?.maxAttempts}
      />

      <ActionButtons
        canCancel={status?.status === 'generating'}
        canExtendTime={status?.timeRemaining < 5000}
        onCancel={() => cancelGeneration(sessionId)}
        onExtendTime={() => extendTimeout(sessionId)}
      />
    </div>
  );
};
```

7.3 Results Display Interface

ResultsDashboard Component

tsx

```tsx
interface ResultsDashboardProps {
  result: GeneratedTicketSet;
  onRegenerate: () => void;
  onDownload: (format: 'pdf' | 'csv' | 'json') => void;
}

const ResultsDashboard: React.FC<ResultsDashboardProps> = ({
  result, onRegenerate, onDownload
}) => {
  return (
    <div className="results-dashboard">
      <ResultsHeader
        quality={result.quality.overall}
        generationTime={result.generation.timeElapsed}
        ticketCount={result.tickets.length}
      />

      <QualityMetricsPanel
        overall={result.quality.overall}
        patterns={result.quality.patternGrouping}
        numbers={result.quality.numberAnalysis}
        diversity={result.quality.diversity}
      />

      <PatternAnalysisSection
        patternGrouping={result.quality.patternGrouping}
        hotPatterns={result.patterns.hotPatterns}
        showDetailed={true}
      />

      <TicketDisplayGrid
        tickets={result.tickets}
        highlightPatterns={true}
        groupByPattern={false}
        showQualityScores={true}
      />

      <PerformanceProjection
        expected={result.expectedPerformance}
        historical={result.patterns.historical}
        showFinancialAnalysis={true}
      />

      <ActionPanel
        onRegenerate={onRegenerate}
        onDownload={onDownload}
```

```
        onShare={() => shareResults(result)}
        onSaveSession={() => saveSession(result)}
      />
    </div>
  );
};
```

---

8. Database Schema {#database-schema}

8.1 Core Tables

sql

```sql
-- Lottery Configurations
CREATE TABLE lottery_configurations (
    id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    name VARCHAR(100) NOT NULL,
    number_min INTEGER NOT NULL DEFAULT 1,
    number_max INTEGER NOT NULL,
    draw_size INTEGER NOT NULL,
    pattern_length INTEGER NOT NULL,
    tier3_prize DECIMAL(10,2) NOT NULL,
    tier4_prize DECIMAL(10,2) NOT NULL,
    tier5_prize DECIMAL(10,2) NOT NULL,
    created_at TIMESTAMP DEFAULT NOW(),
    updated_at TIMESTAMP DEFAULT NOW()
);

-- User Sessions
CREATE TABLE user_sessions (
    id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    session_id VARCHAR(100) UNIQUE NOT NULL,
    lottery_config_id UUID NOT NULL REFERENCES lottery_configurations(id),
    target_tier VARCHAR(10) NOT NULL,
    number_of_tickets INTEGER NOT NULL,
    strategy VARCHAR(20) NOT NULL,
    budget DECIMAL(10,2),
    preferences JSONB DEFAULT '{}',
    status VARCHAR(20) DEFAULT 'active',
    created_at TIMESTAMP DEFAULT NOW(),
    updated_at TIMESTAMP DEFAULT NOW()
);

-- Generated Ticket Sets
CREATE TABLE generated_ticket_sets (
    id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    session_id UUID NOT NULL REFERENCES user_sessions(id),
    status VARCHAR(20) NOT NULL,
    generation_attempts INTEGER NOT NULL,
    generation_time_ms INTEGER NOT NULL,
    stopping_reason VARCHAR(50) NOT NULL,
    overall_quality DECIMAL(5,3) NOT NULL,
    quality_grade VARCHAR(5) NOT NULL,
    pattern_hot_coverage DECIMAL(5,3) NOT NULL,
    pattern_warm_coverage DECIMAL(5,3) NOT NULL,
    pattern_cold_coverage DECIMAL(5,3) NOT NULL,
    pattern_efficiency DECIMAL(5,3) NOT NULL,
    expected_tier3_wins DECIMAL(8,4),
    expected_tier4_wins DECIMAL(8,4),
```

```sql
    expected_tier5_wins DECIMAL(8,6),
    created_at TIMESTAMP DEFAULT NOW()
);

-- Individual Tickets
CREATE TABLE lottery_tickets (
    id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    ticket_set_id UUID NOT NULL REFERENCES generated_ticket_sets(id),
    ticket_index INTEGER NOT NULL,
    numbers INTEGER[] NOT NULL,
    pattern VARCHAR(10) NOT NULL,
    pattern_group VARCHAR(10) NOT NULL,
    generation_strategy VARCHAR(50) NOT NULL,
    quality_score DECIMAL(5,3) NOT NULL,
    expected_matches DECIMAL(4,2),
    created_at TIMESTAMP DEFAULT NOW(),
    CONSTRAINT valid_numbers CHECK (array_length(numbers, 1) = 5),
    CONSTRAINT valid_pattern_group CHECK (pattern_group IN ('hot', 'warm', 'cold', 'unknown'))
);

-- Performance Tracking
CREATE TABLE generation_performance (
    id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    session_id UUID NOT NULL REFERENCES user_sessions(id),
    attempt_number INTEGER NOT NULL,
    quality_score DECIMAL(5,3) NOT NULL,
    pattern_hot_coverage DECIMAL(5,3) NOT NULL,
    pattern_diversity_score DECIMAL(5,3) NOT NULL,
    time_elapsed_ms INTEGER NOT NULL,
    threshold_status JSONB DEFAULT '{}',
    created_at TIMESTAMP DEFAULT NOW()
);
```

8.2 Indexes and Performance

```sql
```

```
-- Performance indexes
CREATE INDEX idx_user_sessions_session_id ON user_sessions(session_id);
CREATE INDEX idx_user_sessions_created_at ON user_sessions(created_at);
CREATE INDEX idx_generated_ticket_sets_session_id ON generated_ticket_sets(session_id);
CREATE INDEX idx_lottery_tickets_ticket_set_id ON lottery_tickets(ticket_set_id);
CREATE INDEX idx_lottery_tickets_pattern ON lottery_tickets(pattern);
CREATE INDEX idx_lottery_tickets_pattern_group ON lottery_tickets(pattern_group);
CREATE INDEX idx_pattern_groups_lottery_config ON pattern_groups(lottery_config_id);
CREATE INDEX idx_pattern_groups_group_type ON pattern_groups(group_type);
CREATE INDEX idx_hot_numbers_lottery_config ON hot_numbers(lottery_config_id);

-- Composite indexes for analytics
CREATE INDEX idx_tickets_pattern_quality ON lottery_tickets(pattern, quality_score DESC);
CREATE INDEX idx_performance_session_attempt ON generation_performance(session_id, attempt_number);
```

---

9. Configuration Management {#configuration}

9.1 Configuration Structure

yaml

```yaml
# config/lottery-engine.yml
lottery_engine:
  version: "2.1.0"

  # Pattern grouping configuration
  pattern_grouping:
    update_frequency: "daily"
    minimum_historical_draws: 1000
    confidence_threshold: 0.95
    hot_pattern_minimum_frequency: 50

    efficiency_multipliers:
      hot: 2.11
      warm: 1.5
      cold: 0.8
      unknown: 1.0

  # Generation engine settings
  generation:
    default_timeout_ms: 30000
    max_timeout_ms: 120000
    progress_report_interval: 1000
    max_concurrent_sessions: 1000

    strategy_defaults:
      conservative:
        diversity_weight: 1.3
        concentration_limit: 0.8
        hot_pattern_bonus: 0.9
      balanced:
        diversity_weight: 1.0
        concentration_limit: 1.0
        hot_pattern_bonus: 1.0
      aggressive:
        diversity_weight: 0.7
        concentration_limit: 1.2
        hot_pattern_bonus: 1.1

  # Stopping criteria thresholds
  stopping_criteria:
    tier3:
      pattern_grouping:
        hot_coverage_minimum: 0.70
        warm_coverage_maximum: 0.20
        cold_coverage_maximum: 0.10
        diversity_minimum_unique_hot: 4
```

```yaml
      efficiency_minimum: 0.65
    traditional:
      optimization_score_minimum: 0.65
      hot_number_coverage_minimum: 0.70
    meta:
      max_attempts_base: 100
      timeout_ms_base: 15000
      stall_threshold: 20
      minimum_quality_gate: 0.60

  tier4:
    pattern_grouping:
      hot_coverage_minimum: 0.85
      warm_coverage_maximum: 0.12
      cold_coverage_maximum: 0.03
      diversity_minimum_unique_hot: 3
      efficiency_minimum: 0.75
    traditional:
      optimization_score_minimum: 0.70
      hot_number_coverage_minimum: 0.80
    meta:
      max_attempts_base: 200
      timeout_ms_base: 25000
      stall_threshold: 30
      minimum_quality_gate: 0.60

  tier5:
    pattern_grouping:
      hot_coverage_minimum: 0.95
      warm_coverage_maximum: 0.05
      cold_coverage_maximum: 0.00
      diversity_minimum_unique_hot: 2
      efficiency_minimum: 0.85
    traditional:
      optimization_score_minimum: 0.75
      hot_number_coverage_minimum: 0.90
    meta:
      max_attempts_base: 300
      timeout_ms_base: 45000
      stall_threshold: 50
      minimum_quality_gate: 0.65

# Environment-specific overrides
environments:
  development:
    generation:
      default_timeout_ms: 10000
```

```
    max_concurrent_sessions: 10

production:
  generation:
    default_timeout_ms: 30000
    max_concurrent_sessions: 1000

testing:
  generation:
    default_timeout_ms: 5000
    max_concurrent_sessions: 5
```

9.2 Dynamic Configuration API

```python
```

```python
class ConfigurationManager:
    def __init__(self):
        self.config_cache = {}
        self.last_update = {}

    def get_stopping_criteria(
        self, target_tier: str, strategy: str,
        lottery_config: str
    ) -> StoppingCriteria:

        # Get base configuration
        base_config = self.get_base_config(target_tier)

        # Apply strategy modifications
        strategy_modifiers = self.get_strategy_modifiers(strategy)
        modified_config = self.apply_strategy_modifiers(
            base_config, strategy_modifiers
        )

        # Apply any dynamic adjustments
        final_config = self.apply_dynamic_adjustments(
            modified_config, lottery_config
        )

        return final_config

    def update_pattern_groups(
        self, lottery_config: str,
        analysis_results: PatternAnalysisResults
    ):
        """Update pattern groupings based on fresh analysis"""

        # Validate analysis results
        if not self.validate_analysis_results(analysis_results):
            raise ValueError("Invalid analysis results")

        # Update database
        self.database.update_pattern_groups(lottery_config, analysis_results)

        # Clear relevant caches
        self.clear_pattern_cache(lottery_config)

        # Log update
        logger.info(f"Updated pattern groups for {lottery_config}")

    def get_dynamic_thresholds(
```

```
    self, target_tier: str, recent_performance: dict
) -> dict:
    """Adjust thresholds based on recent system performance"""

    base_thresholds = self.get_base_thresholds(target_tier)

    # If system is consistently failing to meet thresholds,
    # slightly lower them to improve success rate
    if recent_performance.get('success_rate', 1.0) < 0.7:
        adjustment_factor = 0.95  # Lower thresholds by 5%

        for category in base_thresholds:
            for threshold in base_thresholds[category]:
                if 'minimum' in base_thresholds[category][threshold]:
                    base_thresholds[category][threshold]['minimum'] *= adjustment_factor

    return base_thresholds
```

---

10. Error Handling & Edge Cases {#error-handling}

10.1 Error Classification

python

```python
class LotteryEngineError(Exception):
    """Base exception for lottery engine errors"""
    def __init__(self, message: str, error_code: int, details: dict = None):
        self.message = message
        self.error_code = error_code
        self.details = details or {}
        super().__init__(self.message)

class ValidationError(LotteryEngineError):
    """Input validation errors"""
    pass

class GenerationError(LotteryEngineError):
    """Number generation errors"""
    pass

class PatternError(LotteryEngineError):
    """Pattern analysis errors"""
    pass

class TimeoutError(LotteryEngineError):
    """Generation timeout errors"""
    pass

# Error codes
ERROR_CODES = {
    # Validation errors (4000-4099)
    'INVALID_TIER': 4001,
    'INVALID_TICKET_COUNT': 4002,
    'INVALID_STRATEGY': 4003,
    'INVALID_BUDGET': 4004,
    'INVALID_LOTTERY_CONFIG': 4005,

    # Generation errors (4100-4199)
    'GENERATION_FAILED': 4101,
    'PATTERN_THRESHOLD_NOT_MET': 4102,
    'QUALITY_THRESHOLD_NOT_MET': 4103,
    'INSUFFICIENT_PATTERN_DATA': 4104,

    # System errors (5000-5099)
    'GENERATION_TIMEOUT': 5001,
    'DATABASE_ERROR': 5002,
    'CONFIGURATION_ERROR': 5003,
    'RESOURCE_EXHAUSTION': 5004
}
```

10.2 Edge Case Handling

python

```python
class EdgeCaseHandler:
    def handle_insufficient_hot_patterns(
        self, request: UserRequest, available_patterns: int
    ) -> GenerationStrategy:
        """Handle when there aren't enough hot patterns available"""

        if available_patterns < 2:
            # Fallback to warm patterns
            logger.warning(f"Insufficient hot patterns ({available_patterns}), using warm patterns")
            return self.create_fallback_strategy(request, 'warm_pattern_focus')

        elif available_patterns < request.numberOfTickets * 0.1:
            # Reduce diversity requirements
            logger.info(f"Limited hot patterns, reducing diversity requirements")
            return self.create_reduced_diversity_strategy(request)

        else:
            # Continue with normal strategy
            return self.create_normal_strategy(request)

    def handle_generation_timeout(
        self, session_id: str, best_result: GeneratedTicketSet
    ) -> GeneratedTicketSet:
        """Handle generation timeout with best available result"""

        if best_result.quality.overall.optimizationScore >= 0.60:
            # Accept result if it meets minimum quality gate
            best_result.status = 'partial'
            best_result.generation.stoppingReason = 'timeout'

            logger.info(f"Session {session_id} timed out, accepting partial result")
            return best_result

        else:
            # Generate emergency fallback result
            logger.warning(f"Session {session_id} timed out with poor quality, generating fallback")
            return self.generate_emergency_fallback(session_id)

    def handle_pattern_data_corruption(
        self, lottery_config: str, error: Exception
    ) -> PatternGroupDefinition:
        """Handle corrupted or missing pattern data"""

        logger.error(f"Pattern data corruption for {lottery_config}: {error}")

        # Use conservative fallback patterns
```

```
        fallback_patterns = self.get_conservative_fallback_patterns(lottery_config)

        # Queue pattern data rebuild
        self.queue_pattern_rebuild(lottery_config)

        return fallback_patterns

    def handle_high_concurrency(self, active_sessions: int) -> dict:
        """Handle high system load"""

        if active_sessions > 800:
            # Reduce generation complexity
            return {
                'max_attempts_multiplier': 0.7,
                'timeout_multiplier': 0.8,
                'quality_threshold_relaxation': 0.95
            }
        elif active_sessions > 500:
            return {
                'max_attempts_multiplier': 0.85,
                'timeout_multiplier': 0.9,
                'quality_threshold_relaxation': 0.98
            }
        else:
            return {
                'max_attempts_multiplier': 1.0,
                'timeout_multiplier': 1.0,
                'quality_threshold_relaxation': 1.0
            }
```

10.3 Graceful Degradation

python

```python
class GracefulDegradationManager:
    def __init__(self):
        self.degradation_levels = [
            'normal',          # Full functionality
            'reduced_quality',  # Lower quality thresholds
            'simplified_algo',  # Simpler algorithms
            'basic_random',     # Basic random generation
            'emergency'         # Emergency mode
        ]
        self.current_level = 'normal'

    def assess_system_health(self) -> str:
        """Determine appropriate degradation level"""

        # Check system metrics
        cpu_usage = self.get_cpu_usage()
        memory_usage = self.get_memory_usage()
        active_sessions = self.get_active_sessions()
        db_latency = self.get_db_latency()

        # Determine degradation level
        if cpu_usage > 90 or memory_usage > 85 or db_latency > 1000:
            return 'emergency'
        elif cpu_usage > 80 or memory_usage > 75 or active_sessions > 900:
            return 'basic_random'
        elif cpu_usage > 70 or memory_usage > 65 or active_sessions > 700:
            return 'simplified_algo'
        elif cpu_usage > 60 or memory_usage > 55 or active_sessions > 500:
            return 'reduced_quality'
        else:
            return 'normal'

    def apply_degradation(self, level: str, request: UserRequest) -> UserRequest:
        """Apply degradation modifications to request"""

        if level == 'reduced_quality':
            # Lower quality thresholds by 10%
            if hasattr(request, 'preferences'):
                request.preferences.qualityThreshold *= 0.9

        elif level == 'simplified_algo':
            # Use simpler generation strategies
            request.strategy = 'balanced'  # Force balanced strategy
            request.numberOfTickets = min(request.numberOfTickets, 100)

        elif level == 'basic_random':
```

```
    # Use basic random generation with hot number bias
    request.strategy = 'conservative'
    request.numberOfTickets = min(request.numberOfTickets, 50)

elif level == 'emergency':
    # Minimal functionality
    request.strategy = 'conservative'
    request.numberOfTickets = min(request.numberOfTickets, 20)

return request
```

---

11. Performance Requirements {#performance}

11.1 Performance Targets

yaml

```yaml
performance_targets:
  response_times:
    api_endpoints:
      generate_tickets:
        p50: 15000ms  # 15 seconds
        p95: 30000ms  # 30 seconds
        p99: 45000ms  # 45 seconds
      pattern_analysis:
        p50: 200ms
        p95: 500ms
        p99: 1000ms
      generation_status:
        p50: 50ms
        p95: 100ms
        p99: 200ms

  throughput:
    concurrent_generations: 100  # Simultaneous generation sessions
    requests_per_second: 500     # API requests per second
    database_queries_per_second: 2000

  resource_utilization:
    cpu_usage_max: 80%
    memory_usage_max: 70%
    database_connections_max: 200
    cache_hit_ratio_min: 95%

  scalability:
    users_supported: 10000       # Concurrent active users
    sessions_per_hour: 50000     # Peak session creation rate
    data_retention_days: 365     # Session and performance data

  availability:
    uptime_target: 99.9%         # 8.76 hours downtime per year
    recovery_time_objective: 15min
    recovery_point_objective: 5min
```

11.2 Performance Monitoring

python

```python
class PerformanceMonitor:
    def __init__(self):
        self.metrics = {}
        self.alerts = []

    def track_generation_performance(
        self, session_id: str,
        start_time: float,
        end_time: float,
        attempts: int,
        quality_score: float
    ):
        """Track performance metrics for generation sessions"""

        duration = (end_time - start_time) * 1000  # Convert to milliseconds

        # Record metrics
        self.record_metric('generation.duration', duration)
        self.record_metric('generation.attempts', attempts)
        self.record_metric('generation.quality', quality_score)
        self.record_metric('generation.throughput', 1)

        # Check performance thresholds
        if duration > 30000:  # 30 seconds
            self.create_alert('SLOW_GENERATION', {
                'session_id': session_id,
                'duration': duration,
                'attempts': attempts
            })

        if attempts > 200:
            self.create_alert('HIGH_ATTEMPTS', {
                'session_id': session_id,
                'attempts': attempts,
                'quality': quality_score
            })

    def track_pattern_analysis_performance(
        self, operation: str,
        duration: float,
        cache_hit: bool
    ):
        """Track pattern analysis performance"""

        self.record_metric(f'pattern.{operation}.duration', duration)
        self.record_metric(f'pattern.{operation}.cache_hit', 1 if cache_hit else 0)
```

```
        if duration > 1000 and not cache_hit:  # 1 second for non-cached
            self.create_alert('SLOW_PATTERN_ANALYSIS', {
                'operation': operation,
                'duration': duration,
                'cache_hit': cache_hit
            })

    def get_performance_dashboard(self) -> dict:
        """Generate performance dashboard data"""

        return {
            'current_load': {
                'active_sessions': self.get_active_sessions(),
                'cpu_usage': self.get_cpu_usage(),
                'memory_usage': self.get_memory_usage(),
                'database_connections': self.get_db_connections()
            },
            'generation_metrics': {
                'avg_duration': self.get_avg_metric('generation.duration'),
                'success_rate': self.calculate_success_rate(),
                'avg_quality': self.get_avg_metric('generation.quality'),
                'throughput_per_hour': self.get_hourly_throughput()
            },
            'pattern_metrics': {
                'cache_hit_ratio': self.calculate_cache_hit_ratio(),
                'avg_analysis_time': self.get_avg_metric('pattern.analysis.duration'),
                'update_frequency': self.get_pattern_update_frequency()
            },
            'alerts': {
                'active_alerts': len([a for a in self.alerts if a['active']]),
                'recent_alerts': self.get_recent_alerts(hours=24)
            }
        }
```

---

12. Testing Specifications {#testing}

12.1 Unit Test Requirements

python

```python
# tests/test_pattern_grouping_engine.py
class TestPatternGroupingEngine:
    def test_pattern_extraction(self):
        """Test pattern extraction from number arrays"""
        engine = PatternGroupingEngine(lottery_config_5_35)

        # Test cases
        test_cases = [
            ([1, 15, 23, 28, 34], "01223"),
            ([5, 10, 16, 20, 25], "01122"),
            ([8, 13, 19, 24, 30], "01123"),
            ([2, 7, 11, 17, 22], "00112")
        ]

        for numbers, expected_pattern in test_cases:
            result = engine.extract_pattern(numbers)
            assert result == expected_pattern, f"Expected {expected_pattern}, got {result}"

    def test_pattern_classification(self):
        """Test pattern classification into hot/warm/cold groups"""
        engine = PatternGroupingEngine(lottery_config_5_35)

        # Test hot patterns
        hot_patterns = ["01123", "01223", "01122", "00123", "00122", "00112"]
        for pattern in hot_patterns:
            group = engine.determine_pattern_group(pattern)
            assert group == "hot", f"Pattern {pattern} should be hot, got {group}"

        # Test warm patterns
        warm_patterns = ["11223", "01233", "01222", "00223", "00113", "01112"]
        for pattern in warm_patterns:
            group = engine.determine_pattern_group(pattern)
            assert group == "warm", f"Pattern {pattern} should be warm, got {group}"

    def test_ticket_set_analysis(self):
        """Test analysis of complete ticket sets"""
        engine = PatternGroupingEngine(lottery_config_5_35)

        # Create test ticket set with known pattern distribution
        tickets = [
            LotteryTicket(numbers=[1,15,23,28,34], pattern="01223", patternGroup="hot"),
            LotteryTicket(numbers=[5,10,16,20,25], pattern="01122", patternGroup="hot"),
            LotteryTicket(numbers=[2,17,24,29,33], pattern="01223", patternGroup="warm"),
        ]

        analysis = engine.analyze_ticket_set_patterns(tickets)
```

```
    assert analysis.hotPatternCoverage == 2/3  # 2 out of 3 tickets
    assert analysis.warmPatternCoverage == 1/3  # 1 out of 3 tickets
    assert analysis.patternDiversity.uniqueHotPatterns == 2  # 2 unique hot patterns
```

12.2 Integration Test Requirements

python

```python
# tests/test_generation_integration.py
class TestGenerationIntegration:
    def test_full_generation_flow(self):
        """Test complete generation flow from request to result"""

        request = UserRequest(
            targetTier="tier4",
            numberOfTickets=100,
            strategy="aggressive",
            budget=500,
            lotteryConfig="5_35"
        )

        # Execute generation
        engine = MasterGenerationEngine()
        result = engine.generate_tickets(request)

        # Verify result structure
        assert result.status in ['success', 'partial', 'fallback']
        assert len(result.tickets) == 100
        assert result.quality.overall.optimizationScore >= 0.60

        # Verify pattern grouping requirements for aggressive tier4
        assert result.quality.patternGrouping.hotPatternCoverage >= 0.85
        assert result.quality.patternGrouping.coldPatternCoverage <= 0.03

    def test_stopping_criteria_enforcement(self):
        """Test that stopping criteria are properly enforced"""

        request = UserRequest(
            targetTier="tier3",
            numberOfTickets=50,
            strategy="balanced",
            budget=200,
            lotteryConfig="5_35"
        )

        engine = MasterGenerationEngine()
        result = engine.generate_tickets(request)

        # Verify tier3 balanced criteria
        criteria = engine.calculate_stopping_criteria(request)
        quality = result.quality

        if result.status == 'success':
            # Should meet pattern grouping thresholds
```

```
        assert quality.patternGrouping.hotPatternCoverage >= criteria.patternGrouping.hotPatternCoverage.minim
        assert quality.patternGrouping.warmPatternCoverage <= criteria.patternGrouping.warmPatternLimit.maxim

    def test_performance_within_limits(self):
        """Test that generation completes within performance limits"""

        request = UserRequest(
            targetTier="tier4",
            numberOfTickets=200,
            strategy="aggressive",
            budget=1000,
            lotteryConfig="5_35"
        )

        start_time = time.time()
        engine = MasterGenerationEngine()
        result = engine.generate_tickets(request)
        end_time = time.time()

        duration = (end_time - start_time) * 1000  # milliseconds

        # Should complete within 45 seconds for 200 tickets
        assert duration <= 45000, f"Generation took {duration}ms, expected <= 45000ms"
        assert result.generation.attempts <= 400, f"Used {result.generation.attempts} attempts, expected <= 400"
```

12.3 Load Test Specifications

python

```python
# tests/test_load_performance.py
class TestLoadPerformance:
    def test_concurrent_generation_sessions(self):
        """Test system under concurrent load"""

        import concurrent.futures
        import threading

        def generate_tickets(session_id):
            request = UserRequest(
                sessionId=f"load_test_{session_id}",
                targetTier="tier3",
                numberOfTickets=50,
                strategy="balanced",
                budget=200,
                lotteryConfig="5_35"
            )

            engine = MasterGenerationEngine()
            start_time = time.time()
            result = engine.generate_tickets(request)
            end_time = time.time()

            return {
                'session_id': session_id,
                'duration': (end_time - start_time) * 1000,
                'status': result.status,
                'quality': result.quality.overall.optimizationScore,
                'attempts': result.generation.attempts
            }

        # Run 50 concurrent generation sessions
        with concurrent.futures.ThreadPoolExecutor(max_workers=50) as executor:
            futures = [executor.submit(generate_tickets, i) for i in range(50)]
            results = [future.result() for future in concurrent.futures.as_completed(futures)]

        # Analyze results
        successful_sessions = [r for r in results if r['status'] == 'success']
        avg_duration = sum(r['duration'] for r in results) / len(results)
        max_duration = max(r['duration'] for r in results)

        # Assertions
        assert len(successful_sessions) >= 45, f"Only {len(successful_sessions)}/50 sessions succeeded"
        assert avg_duration <= 20000, f"Average duration {avg_duration}ms exceeded 20s limit"
        assert max_duration <= 35000, f"Max duration {max_duration}ms exceeded 35s limit"
```

```python
def test_pattern_cache_performance(self):
    """Test pattern analysis cache performance under load"""

    engine = PatternGroupingEngine(lottery_config_5_35)

    # Warm up cache
    engine.load_pattern_groups()

    # Measure cache hit performance
    start_time = time.time()
    for _ in range(1000):
        engine.get_pattern_efficiency("01223")
    end_time = time.time()

    avg_cache_hit_time = (end_time - start_time) / 1000 * 1000  # microseconds

    assert avg_cache_hit_time <= 1.0, f"Cache hit time {avg_cache_hit_time}μs too slow"
```

---

13. Deployment & Operations {#deployment}

13.1 Docker Configuration

dockerfile

```dockerfile
# Dockerfile
FROM python:3.11-slim

# Set working directory
WORKDIR /app

# Install system dependencies
RUN apt-get update && apt-get install -y \
    gcc \
    g++ \
    libpq-dev \
    && rm -rf /var/lib/apt/lists/*

# Copy requirements and install Python dependencies
COPY requirements.txt .
RUN pip install --no-cache-dir -r requirements.txt

# Copy application code
COPY . .

# Create non-root user
RUN groupadd -r lottery && useradd -r -g lottery lottery
RUN chown -R lottery:lottery /app
USER lottery

# Health check
HEALTHCHECK --interval=30s --timeout=10s --start-period=30s --retries=3 \
    CMD python -c "import requests; requests.get('http://localhost:8000/health').raise_for_status()"

# Expose port
EXPOSE 8000

# Run application
CMD ["gunicorn", "--bind", "0.0.0.0:8000", "--workers", "4", "--worker-class", "gevent", "app:app"]
```

yaml

```yaml
# docker-compose.yml
version: '3.8'
services:
  lottery-engine:
    build: .
    ports:
      - "8000:8000"
    environment:
      - DATABASE_URL=postgresql://lottery:password@db:5432/lottery_engine
      - REDIS_URL=redis://redis:6379/0
      - ENVIRONMENT=production
    depends_on:
      - db
      - redis
    volumes:
      - ./config:/app/config
      - ./logs:/app/logs
    restart: unless-stopped

  db:
    image: postgres:15
    environment:
      POSTGRES_DB: lottery_engine
      POSTGRES_USER: lottery
      POSTGRES_PASSWORD: password
    volumes:
      - postgres_data:/var/lib/postgresql/data
      - ./init.sql:/docker-entrypoint-initdb.d/init.sql
    restart: unless-stopped

  redis:
    image: redis:7-alpine
    volumes:
      - redis_data:/data
    restart: unless-stopped

  nginx:
    image: nginx:alpine
    ports:
      - "80:80"
      - "443:443"
    volumes:
      - ./nginx.conf:/etc/nginx/nginx.conf
      - ./ssl:/etc/nginx/ssl
    depends_on:
      - lottery-engine
```

```
    restart: unless-stopped

volumes:
  postgres_data:
  redis_data:
```

13.2 Kubernetes Deployment

yaml

```yaml
# k8s/deployment.yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: lottery-engine
  labels:
    app: lottery-engine
spec:
  replicas: 3
  selector:
    matchLabels:
      app: lottery-engine
  template:
    metadata:
      labels:
        app: lottery-engine
    spec:
      containers:
      - name: lottery-engine
        image: lottery-engine:latest
        ports:
        - containerPort: 8000
        env:
        - name: DATABASE_URL
          valueFrom:
            secretKeyRef:
              name: lottery-secrets
              key: database-url
        - name: REDIS_URL
          valueFrom:
            configMapKeyRef:
              name: lottery-config
              key: redis-url
        resources:
          requests:
            memory: "512Mi"
            cpu: "250m"
          limits:
            memory: "2Gi"
            cpu: "1000m"
        livenessProbe:
          httpGet:
            path: /health
            port: 8000
          initialDelaySeconds: 30
          periodSeconds: 30
```

```
      readinessProbe:
        httpGet:
          path: /ready
          port: 8000
        initialDelaySeconds: 5
        periodSeconds: 5
---
apiVersion: v1
kind: Service
metadata:
  name: lottery-engine-service
spec:
  selector:
    app: lottery-engine
  ports:
    - protocol: TCP
      port: 80
      targetPort: 8000
  type: LoadBalancer
```

13.3 Monitoring & Observability

yaml

```
# monitoring/prometheus.yml
global:
  scrape_interval: 15s

scrape_configs:
  - job_name: 'lottery-engine'
    static_configs:
      - targets: ['lottery-engine:8000']
    metrics_path: '/metrics'
    scrape_interval: 10s

rule_files:
  - "alert_rules.yml"

alerting:
  alertmanagers:
    - static_configs:
        - targets:
          - alertmanager:9093
```

yaml

```yaml
# monitoring/alert_rules.yml
groups:
- name: lottery-engine-alerts
  rules:
  - alert: HighGenerationLatency
    expr: histogram_quantile(0.95, rate(generation_duration_seconds_bucket[5m])) > 30
    for: 2m
    labels:
      severity: warning
    annotations:
      summary: "High generation latency detected"
      description: "95th percentile latency is {{ $value }} seconds"

  - alert: LowSuccessRate
    expr: rate(generation_success_total[5m]) / rate(generation_total[5m]) < 0.8
    for: 5m
    labels:
      severity: critical
    annotations:
      summary: "Generation success rate below 80%"
      description: "Success rate is {{ $value | humanizePercentage }}"

  - alert: PatternDataStale
    expr: time() - pattern_data_last_update_timestamp > 86400
    for: 0m
    labels:
      severity: warning
    annotations:
      summary: "Pattern data hasn't been updated in 24 hours"
      description: "Last update was {{ $value | humanizeDuration }} ago"
```

13.4 Operational Procedures

```python
```

```python
# ops/health_checks.py
class HealthChecker:
    def __init__(self):
        self.checks = [
            self.check_database_connectivity,
            self.check_redis_connectivity,
            self.check_pattern_data_freshness,
            self.check_generation_capacity,
            self.check_memory_usage
        ]

    def health_check(self) -> dict:
        """Comprehensive health check"""
        results = {}
        overall_healthy = True

        for check in self.checks:
            try:
                check_name = check.__name__.replace('check_', '')
                result = check()
                results[check_name] = result

                if not result.get('healthy', False):
                    overall_healthy = False

            except Exception as e:
                results[check.__name__] = {
                    'healthy': False,
                    'error': str(e),
                    'timestamp': time.time()
                }
                overall_healthy = False

        return {
            'healthy': overall_healthy,
            'checks': results,
            'timestamp': time.time()
        }

    def check_database_connectivity(self) -> dict:
        """Check database connection and query performance"""
        start_time = time.time()

        try:
            # Test basic connectivity
            db.execute("SELECT 1")
```

```
        # Test pattern data availability
        pattern_count = db.execute(
            "SELECT COUNT(*) FROM pattern_groups WHERE group_type = 'hot'"
        ).scalar()

        query_time = (time.time() - start_time) * 1000

        return {
            'healthy': pattern_count > 0 and query_time < 100,
            'pattern_count': pattern_count,
            'query_time_ms': query_time,
            'timestamp': time.time()
        }

    except Exception as e:
        return {
            'healthy': False,
            'error': str(e),
            'timestamp': time.time()
        }

def check_generation_capacity(self) -> dict:
    """Check system capacity for new generation requests"""

    active_sessions = self.get_active_session_count()
    cpu_usage = psutil.cpu_percent(interval=1)
    memory_usage = psutil.virtual_memory().percent

    # Determine if system can handle new requests
    can_accept_requests = (
        active_sessions < 900 and
        cpu_usage < 85 and
        memory_usage < 80
    )

    return {
        'healthy': can_accept_requests,
        'active_sessions': active_sessions,
        'cpu_usage': cpu_usage,
        'memory_usage': memory_usage,
        'capacity_available': can_accept_requests,
        'timestamp': time.time()
    }
```

python

```python
# ops/maintenance.py
class MaintenanceOperations:
    def update_pattern_data(self, lottery_config: str, force: bool = False):
        """Update pattern groupings from latest lottery data"""

        logger.info(f"Starting pattern data update for {lottery_config}")

        try:
            # Check if update is needed
            if not force and not self.is_pattern_update_needed(lottery_config):
                logger.info("Pattern data is current, skipping update")
                return

            # Fetch latest lottery results
            latest_results = self.fetch_latest_lottery_results(lottery_config)

            # Analyze patterns
            analyzer = PatternAnalyzer(lottery_config)
            new_patterns = analyzer.analyze_results(latest_results)

            # Validate new patterns
            if not self.validate_pattern_data(new_patterns):
                raise ValueError("New pattern data failed validation")

            # Update database with transaction
            with db.transaction():
                self.backup_current_patterns(lottery_config)
                self.update_pattern_groups(lottery_config, new_patterns)
                self.update_hot_numbers(lottery_config, new_patterns)
                self.update_proven_combinations(lottery_config, new_patterns)

            # Clear caches
            self.clear_pattern_caches(lottery_config)

            logger.info(f"Successfully updated pattern data for {lottery_config}")

        except Exception as e:
            logger.error(f"Failed to update pattern data: {e}")
            # Rollback if necessary
            self.rollback_pattern_update(lottery_config)
            raise

    def cleanup_old_sessions(self, days_old: int = 7):
        """Clean up old session data"""

        cutoff_date = datetime.now() - timedelta(days=days_old)
```

```
    # Delete old sessions and related data
    with db.transaction():
        deleted_tickets = db.execute("""
            DELETE FROM lottery_tickets
            WHERE ticket_set_id IN (
                SELECT id FROM generated_ticket_sets
                WHERE created_at < %s
            )
        """, (cutoff_date,)).rowcount

        deleted_sets = db.execute("""
            DELETE FROM generated_ticket_sets
            WHERE created_at < %s
        """, (cutoff_date,)).rowcount

        deleted_sessions = db.execute("""
            DELETE FROM user_sessions
            WHERE created_at < %s AND status != 'active'
        """, (cutoff_date,)).rowcount

    logger.info(f"Cleanup completed: {deleted_sessions} sessions, "
                f"{deleted_sets} ticket sets, {deleted_tickets} tickets")

def performance_optimization(self):
    """Run performance optimization tasks"""

    # Update table statistics
    db.execute("ANALYZE pattern_groups")
    db.execute("ANALYZE user_sessions")
    db.execute("ANALYZE generated_ticket_sets")
    db.execute("ANALYZE lottery_tickets")

    # Rebuild critical indexes if needed
    self.rebuild_indexes_if_needed()

    # Update Redis cache with fresh data
    self.warm_redis_cache()

    logger.info("Performance optimization completed")
```

14. Security & Compliance

14.1 Security Requirements

```python
```

```python
# security/input_validation.py
class InputValidator:
    def validate_generation_request(self, request_data: dict) -> dict:
        """Validate and sanitize generation request"""

        errors = []
        cleaned_data = {}

        # Target tier validation
        if 'targetTier' not in request_data:
            errors.append("targetTier is required")
        elif request_data['targetTier'] not in ['tier3', 'tier4', 'tier5']:
            errors.append("targetTier must be tier3, tier4, or tier5")
        else:
            cleaned_data['targetTier'] = request_data['targetTier']

        # Number of tickets validation
        if 'numberOfTickets' not in request_data:
            errors.append("numberOfTickets is required")
        else:
            try:
                tickets = int(request_data['numberOfTickets'])
                if tickets < 1 or tickets > 500:
                    errors.append("numberOfTickets must be between 1 and 500")
                else:
                    cleaned_data['numberOfTickets'] = tickets
            except ValueError:
                errors.append("numberOfTickets must be a valid integer")

        # Strategy validation
        if 'strategy' not in request_data:
            errors.append("strategy is required")
        elif request_data['strategy'] not in ['conservative', 'balanced', 'aggressive']:
            errors.append("strategy must be conservative, balanced, or aggressive")
        else:
            cleaned_data['strategy'] = request_data['strategy']

        # Budget validation
        if 'budget' in request_data:
            try:
                budget = float(request_data['budget'])
                if budget < 1 or budget > 100000:
                    errors.append("budget must be between $1 and $100,000")
                else:
                    cleaned_data['budget'] = budget
            except ValueError:
```

```
            errors.append("budget must be a valid number")

        if errors:
            raise ValidationError("Invalid request data", 4000, {'errors': errors})

        return cleaned_data

    def sanitize_session_id(self, session_id: str) -> str:
        """Sanitize session ID to prevent injection"""

        if not session_id:
            raise ValidationError("Session ID is required", 4001)

        # Must be valid UUID format
        import uuid
        try:
            uuid.UUID(session_id)
            return session_id
        except ValueError:
            raise ValidationError("Invalid session ID format", 4002)
```

14.2 Rate Limiting

python

```python
# security/rate_limiter.py
class RateLimiter:
    def __init__(self, redis_client):
        self.redis = redis_client
        self.limits = {
            'generation_requests': {'count': 10, 'window': 3600},  # 10 per hour
            'api_requests': {'count': 1000, 'window': 3600},       # 1000 per hour
            'status_checks': {'count': 600, 'window': 3600}        # 600 per hour
        }

    def check_rate_limit(self, client_id: str, limit_type: str) -> dict:
        """Check if client is within rate limits"""

        if limit_type not in self.limits:
            raise ValueError(f"Unknown limit type: {limit_type}")

        limit_config = self.limits[limit_type]
        key = f"rate_limit:{limit_type}:{client_id}"

        # Get current count
        current_count = self.redis.get(key)
        if current_count is None:
            current_count = 0
        else:
            current_count = int(current_count)

        # Check if limit exceeded
        if current_count >= limit_config['count']:
            remaining_time = self.redis.ttl(key)
            return {
                'allowed': False,
                'limit': limit_config['count'],
                'current': current_count,
                'reset_in': remaining_time
            }

        # Increment counter
        pipe = self.redis.pipeline()
        pipe.incr(key)
        pipe.expire(key, limit_config['window'])
        pipe.execute()

        return {
            'allowed': True,
            'limit': limit_config['count'],
            'current': current_count + 1,
```

```
        'reset_in': limit_config['window']
    }
```

14.3 Data Privacy

python

```python
# security/privacy.py
class PrivacyManager:
    def anonymize_session_data(self, session_data: dict) -> dict:
        """Remove or anonymize PII from session data"""

        anonymized = session_data.copy()

        # Remove any IP addresses
        anonymized.pop('client_ip', None)

        # Hash session ID for analytics
        if 'session_id' in anonymized:
            anonymized['session_id_hash'] = hashlib.sha256(
                anonymized['session_id'].encode()
            ).hexdigest()[:16]
            anonymized.pop('session_id')

        # Remove any user identifiers
        anonymized.pop('user_id', None)
        anonymized.pop('email', None)

        return anonymized

    def implement_gdpr_deletion(self, session_id: str):
        """Delete all data associated with a session (GDPR compliance)"""

        with db.transaction():
            # Delete tickets
            db.execute("""
                DELETE FROM lottery_tickets
                WHERE ticket_set_id IN (
                    SELECT id FROM generated_ticket_sets
                    WHERE session_id = (
                        SELECT id FROM user_sessions WHERE session_id = %s
                    )
                )
            """, (session_id,))

            # Delete ticket sets
            db.execute("""
                DELETE FROM generated_ticket_sets
                WHERE session_id = (
                    SELECT id FROM user_sessions WHERE session_id = %s
                )
            """, (session_id,))
```

```
        # Delete performance data
        db.execute("""
            DELETE FROM generation_performance
            WHERE session_id = (
                SELECT id FROM user_sessions WHERE session_id = %s
            )
        """, (session_id,))

        # Delete session
        db.execute("""
            DELETE FROM user_sessions WHERE session_id = %s
        """, (session_id,))

    logger.info(f"GDPR deletion completed for session {session_id}")
```

---

15. API Documentation Examples

15.1 OpenAPI Specification

yaml

```yaml
# api/openapi.yml
openapi: 3.0.3
info:
  title: Lottery Prize Tier Targeting API
  description: Advanced lottery number generation with pattern-based optimization
  version: 2.1.0
  contact:
    name: API Support
    email: support@lottery-engine.com

servers:
  - url: https://api.lottery-engine.com/v2
    description: Production server
  - url: https://staging-api.lottery-engine.com/v2
    description: Staging server

paths:
  /generate-tickets:
    post:
      summary: Generate optimized lottery tickets
      description: Generate lottery tickets using pattern-based optimization targeting specific prize tiers
      operationId: generateTickets
      tags:
        - Generation
      requestBody:
        required: true
        content:
          application/json:
            schema:
              $ref: '#/components/schemas/GenerationRequest'
            examples:
              tier4_aggressive:
                summary: Aggressive Tier 4 targeting
                value:
                  targetTier: "tier4"
                  numberOfTickets: 100
                  strategy: "aggressive"
                  budget: 500
                  lotteryType: "5/35"
      responses:
        '200':
          description: Successfully generated tickets
          content:
            application/json:
              schema:
                $ref: '#/components/schemas/GenerationResponse'
```

```yaml
      '400':
        description: Invalid request parameters
        content:
          application/json:
            schema:
              $ref: '#/components/schemas/ErrorResponse'
      '422':
        description: Generation failed to meet criteria
        content:
          application/json:
            schema:
              $ref: '#/components/schemas/GenerationFailedResponse'
      '429':
        description: Rate limit exceeded
        content:
          application/json:
            schema:
              $ref: '#/components/schemas/RateLimitResponse'

components:
  schemas:
    GenerationRequest:
      type: object
      required:
        - targetTier
        - numberOfTickets
        - strategy
        - lotteryType
      properties:
        targetTier:
          type: string
          enum: [tier3, tier4, tier5]
          description: Prize tier to optimize for
        numberOfTickets:
          type: integer
          minimum: 1
          maximum: 500
          description: Number of tickets to generate
        strategy:
          type: string
          enum: [conservative, balanced, aggressive]
          description: Risk/reward strategy
        budget:
          type: number
          minimum: 1
          maximum: 100000
          description: Total budget consideration
```

```yaml
      lotteryType:
        type: string
        example: "5/35"
        description: Lottery format (numbers to pick / total numbers)
      preferences:
        type: object
        properties:
          maxGenerationTime:
            type: integer
            description: Maximum generation time in milliseconds
          qualityThreshold:
            type: number
            minimum: 0.5
            maximum: 1.0
            description: Minimum quality threshold

  GenerationResponse:
    type: object
    properties:
      sessionId:
        type: string
        format: uuid
        description: Unique session identifier
      status:
        type: string
        enum: [success, partial, fallback]
        description: Generation result status
      tickets:
        type: array
        items:
          $ref: '#/components/schemas/LotteryTicket'
      generation:
        $ref: '#/components/schemas/GenerationMetadata'
      quality:
        $ref: '#/components/schemas/QualityMetrics'
      expectedPerformance:
        $ref: '#/components/schemas/PerformanceProjection'
```

---

16. Implementation Checklist

16.1 Development Phases

```markdown
```

```
## Phase 1: Core Foundation (Weeks 1-4)
- [ ] Database schema implementation
- [ ] Pattern grouping data models
- [ ] Basic generation engine structure
- [ ] Hot/warm/cold pattern classification
- [ ] Unit test framework setup
- [ ] Docker development environment

## Phase 2: Pattern Integration (Weeks 5-8)
- [ ] Pattern grouping engine implementation
- [ ] Pattern-to-numbers generation algorithm
- [ ] Stopping criteria engine (pattern-based)
- [ ] Quality evaluation system
- [ ] Integration testing framework
- [ ] Performance baseline establishment

## Phase 3: Advanced Features (Weeks 9-12)
- [ ] Strategy-specific modifications
- [ ] Real-time generation progress
- [ ] Advanced stopping criteria
- [ ] Error handling & fallback systems
- [ ] Load testing implementation
- [ ] Security & rate limiting

## Phase 4: UI & Polish (Weeks 13-16)
- [ ] Frontend interface implementation
- [ ] Real-time progress indicators
- [ ] Results dashboard
- [ ] API documentation
- [ ] Monitoring & alerting
- [ ] Production deployment preparation

## Phase 5: Launch & Operations (Weeks 17-20)
- [ ] Production deployment
- [ ] Performance monitoring setup
- [ ] User acceptance testing
- [ ] Documentation completion
- [ ] Support procedures
- [ ] Success metrics tracking
```

16.2 Success Criteria Validation

```
python
```

```python
# validation/success_criteria.py
class SuccessCriteriaValidator:
    def validate_pattern_grouping_performance(self, results: list) -> dict:
        """Validate pattern grouping delivers promised performance"""

        hot_pattern_results = [r for r in results if r['primary_pattern_group'] == 'hot']

        # Calculate actual efficiency
        total_wins = sum(r['total_wins'] for r in hot_pattern_results)
        total_tickets = sum(r['total_tickets'] for r in hot_pattern_results)
        actual_win_rate = total_wins / total_tickets if total_tickets > 0 else 0

        # Compare to theoretical (2.25% base rate)
        theoretical_rate = 0.0225
        efficiency_multiplier = actual_win_rate / theoretical_rate

        # Validate against promised 2.11x efficiency
        meets_efficiency = efficiency_multiplier >= 2.0  # Allow 5% margin

        return {
            'meets_criteria': meets_efficiency,
            'actual_efficiency': efficiency_multiplier,
            'promised_efficiency': 2.11,
            'margin': efficiency_multiplier - 2.11,
            'confidence_level': self.calculate_confidence(hot_pattern_results)
        }

    def validate_stopping_criteria_effectiveness(self, sessions: list) -> dict:
        """Validate stopping criteria work as intended"""

        successful_sessions = [s for s in sessions if s['status'] == 'success']
        pattern_threshold_stops = [s for s in successful_sessions
                                   if s['stopping_reason'] == 'pattern_thresholds']

        effectiveness_rate = len(pattern_threshold_stops) / len(successful_sessions)
        avg_attempts = sum(s['attempts'] for s in pattern_threshold_stops) / len(pattern_threshold_stops)

        return {
            'pattern_criteria_effectiveness': effectiveness_rate >= 0.8,  # 80% should stop on pattern criteria
            'average_attempts': avg_attempts,
            'success_rate': len(successful_sessions) / len(sessions),
            'meets_performance_targets': avg_attempts <= 50  # Should find solution quickly
        }
```

This completes the comprehensive output specification for implementing the Lottery Prize Tier Targeting System with Pattern String Grouping integration. The specification provides detailed technical requirements, algorithms, APIs, database schemas, testing procedures, and operational guidelines needed to build a production-ready system that delivers the promised 2.11x efficiency improvement through intelligent pattern-based lottery number generation.