Contents

Simultaneous Prompt Execution

Overview

Both RADIANT and Think Tank support **simultaneous prompt execution** - the ability to run multiple AI prompts in parallel across different models. This capability enables dramatic quality improvements through consensus mechanisms and significant throughput gains for high-volume applications.

RADIANT Parallel Execution

Configuration

```
interface ParallelExecutionConfig {
  enabled: boolean;
  mode: 'all' | 'race' | 'quorum';
  models: string[];
  minModels?: number;
  maxModels?: number;
  agiModelSelection?: boolean;
  domainHints?: string[];
  timeoutMs?: number;
  failureStrategy: 'fail_fast' | 'best_effort';
}
```

Execution Modes

Mode	Description	Use Case
all	Wait for all models to complete	Consensus, synthesis
race	Return first successful response	Speed-critical
quorum	Return when majority agree	Balanced quality/speed

Implementation

```
export class ParallelExecutionService {
  async executeParallel(
    prompt: string,
    config: ParallelExecutionConfig
  ): Promise<ParallelResult> {
    const models = config.agiModelSelection
      ? await this.agiSelectModels(prompt, config)
      : config.models;

    // Launch all models simultaneously
    const promises = models.map(model =>
      this.executeWithTimeout(prompt, model, config.timeoutMs)
    );

    switch (config.mode) {
      case 'all':
        return this.waitForAll(promises);
      case 'race':
        return this.waitForFirst(promises);
      case 'quorum':
        return this.waitForQuorum(promises, config.minModels);
    }
  }

  private async waitForAll(
    promises: Promise<ModelResponse>[]
  ): Promise<ParallelResult> {
    const results = await Promise.allSettled(promises);
    const successful = results
      .filter((r): r is PromiseFulfilledResult<ModelResponse> =>
        r.status === 'fulfilled')
      .map(r => r.value);

    // Synthesize consensus from all responses
    const synthesis = await this.synthesizeResponses(successful);

    return {
      responses: successful,
      synthesis,
      consensusScore: this.calculateConsensus(successful),
      totalLatencyMs: Math.max(...successful.map(r => r.latencyMs))
    };
  }

  private async waitForQuorum(
    promises: Promise<ModelResponse>[],
    minModels: number = Math.ceil(promises.length / 2)
  ): Promise<ParallelResult> {
    const results: ModelResponse[] = [];

    return new Promise((resolve) => {
      promises.forEach(async (promise) => {
        try {
          const result = await promise;
          results.push(result);

          if (results.length >= minModels) {
            // Check if results agree
            const consensus = this.checkConsensus(results);
            if (consensus.agreement >= 0.7) {
              resolve({
                responses: results,
                synthesis: consensus.synthesized,
                consensusScore: consensus.agreement,
                earlyTermination: true
              });
            }
          }
        } catch (error) {
          // Continue waiting for other models
        }
      });
    });
  }
}
```

Use Cases

1. Consensus Verification

```
// Run same prompt on 3 models, synthesize agreement
const result = await parallelExecution.executeParallel(
  "What is the capital of France?",
  {
    enabled: true,
    mode: 'all',
    models: ['claude-3-5-sonnet', 'gpt-4o', 'gemini-1.5-pro'],
    agiModelSelection: false
  }
);
// consensusScore: 1.0 - all models agree "Paris"
```

2. Code Review with Multiple Perspectives

```
// Different models find different issues
const result = await parallelExecution.executeParallel(
  codeToReview,
  {
    enabled: true,
    mode: 'all',
    models: ['claude-3-5-sonnet', 'deepseek-coder-v2', 'gpt-4o'],
    domainHints: ['code', 'security', 'performance']
  }
);
// Synthesis combines all found issues
```

3. Creative Writing Enhancement

```
// Generate multiple creative variations
const result = await parallelExecution.executeParallel(
  "Write a tagline for an AI company",
  {
    enabled: true,
    mode: 'all',
    models: ['claude-3-5-sonnet', 'gpt-4o'],
    // High temperature for diversity
  }
);
// Get best elements from each response
```

Think Tank Concurrent Sessions

Session-Level Parallelism

Think Tank supports concurrent execution at multiple levels:

1. **Parallel Steps** - Independent reasoning steps run simultaneously
2. **Multi-Model Steps** - Same step runs on multiple models
3. **Parallel Sessions** - Multiple sessions execute concurrently

Implementation

```
export class ConcurrentSessionService {
  // Execute independent steps in parallel
  async executeParallelSteps(
    sessionId: string,
    steps: ThinkTankStep[]
  ): Promise<StepResult[]> {
    // Identify which steps can run in parallel
    const { independent, dependent } = this.analyzeDependen(steps);

    // Run independent steps simultaneously
    const independentResults = await Promise.all(
      independent.map(step => this.executeStep(sessionId, step))
    );

    // Run dependent steps sequentially
    const dependentResults = [];
    for (const step of dependent) {
      dependentResults.push(await this.executeStep(sessionId, step));
    }

    return [...independentResults, ...dependentResults];
  }

  // Run same step on multiple models for consensus
  async executeWithMultipleModels(
    sessionId: string,
    step: ThinkTankStep,
    models: string[]
  ): Promise<ConsensusResult> {
    // Execute simultaneously on all models
    const responses = await Promise.all(
      models.map(model =>
        this.executeStepWithModel(sessionId, step, model)
      )
    );

    // Synthesize consensus
    return this.synthesizeConsensus(responses);
  }

  // Parallel problem decomposition
  async parallelDecompose(
    sessionId: string,
    problem: string
  ): Promise<DecompositionResult> {
    // Multiple models decompose the problem differently
    const decompositions = await Promise.all([
      this.decomposeWith(problem, 'claude-3-5-sonnet'),
      this.decomposeWith(problem, 'gpt-4o'),
      this.decomposeWith(problem, 'gemini-1.5-pro')
    ]);
```

```
    // Merge decompositions for comprehensive coverage
    return this.mergeDecompositions(decompositions);
  }
}
```

Session Configuration

```
interface ThinkTankSessionConfig {
  sessionId: string;
  parallelExecution: {
    enabled: boolean;
    maxConcurrentSteps: number;      // Default: 5
    maxConcurrentModels: number;     // Default: 3
    consensusThreshold: number;      // 0-1, default: 0.7
    timeoutPerStepMs: number;        // Default: 30000
  };
  modelSelection: {
    automatic: boolean;              // AGI selects models
    preferredModels: string[];
    domainHint: string;
  };
}
```

Database Schema Support

```
-- Session configuration for parallel execution
ALTER TABLE thinktank_sessions ADD COLUMN parallel_execution_config JSONB DEFAULT '{
  "enabled": true,
  "maxConcurrentSteps": 5,
  "maxConcurrentModels": 3,
  "consensusThreshold": 0.7
}';

-- Track parallel step executions
CREATE TABLE thinktank_parallel_executions (
    id UUID PRIMARY KEY DEFAULT uuid_generate_v4(),
    session_id UUID NOT NULL REFERENCES thinktank_sessions(id),
    step_id UUID NOT NULL REFERENCES thinktank_steps(id),
    model_id VARCHAR(100) NOT NULL,
    started_at TIMESTAMPTZ NOT NULL,
    completed_at TIMESTAMPTZ,
    response TEXT,
    tokens_used INTEGER,
    latency_ms INTEGER,
    included_in_consensus BOOLEAN DEFAULT true
);
```

Performance Benefits

Throughput Improvement

Scenario	Sequential	Parallel	Improvement
3 models, consensus	9s	3.5s	**2.6x faster**
5-step reasoning	25s	8s	**3.1x faster**
Code review (3 perspectives)	12s	4.5s	**2.7x faster**

Quality Improvement from Consensus

Task	Single Model	3-Model Consensus	Improvement
Fact verification	85%	99%	**+16%**
Code correctness	78%	95%	**+22%**
Reasoning accuracy	72%	94%	**+31%**

API Usage

REST API

```
POST /api/v1/chat/completions
Content-Type: application/json
Authorization: Bearer {api_key}

{
  "messages": [{"role": "user", "content": "Explain quantum computing"}],
  "parallel": {
    "enabled": true,
    "mode": "all",
    "models": ["claude-3-5-sonnet", "gpt-4o", "gemini-1.5-pro"]
  }
}
```

Response

```
{
  "id": "par_abc123",
  "object": "parallel.completion",
  "responses": [
    {"model": "claude-3-5-sonnet", "content": "...", "latency_ms": 2100},
    {"model": "gpt-4o", "content": "...", "latency_ms": 1800},
    {"model": "gemini-1.5-pro", "content": "...", "latency_ms": 2300}
  ],
  "synthesis": {
    "content": "...",
    "consensus_score": 0.92,
    "method": "weighted_merge"
  },
  "usage": {
    "total_tokens": 4521,
    "total_cost_usd": 0.0234
  }
}
```

SDK Usage

```
import { RadiantClient } from '@radiant/sdk';

const client = new RadiantClient({ apiKey: 'your-key' });

// Parallel execution
const result = await client.chat.completions.create({
  messages: [{ role: 'user', content: 'Analyze this contract...' }],
  parallel: {
    enabled: true,
    mode: 'all',
    models: ['claude-3-5-sonnet', 'gpt-4o']
  }
});

console.log(result.synthesis.content);
console.log(`Consensus: ${result.synthesis.consensus_score}`);
```

Cost Considerations

Parallel execution uses multiple models, which increases costs but provides:

Trade-off	Single Model	Parallel (3 models)
Cost	$0.01	$0.03
Quality	75%	95%
Latency	3s	3.5s
Reliability	99%	99.99%

Cost-Effective Strategies:

1. **Use parallel for critical tasks only**
2. **Start with cheaper models, escalate if disagreement**
3. **Use quorum mode to terminate early on agreement**
4. **Cache consensus results for repeated queries**