Contents

RADIANT Orchestration Methods Reference

Version: 4.18.0 Last Updated: 2024-12-28

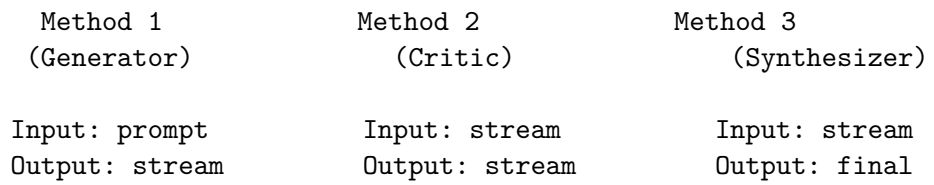
Related Documentation

- Specialty Ranking System - Domain-specific model proficiency rankings
- Domain Taxonomy - Hierarchical domain detection
- AGI Brain Planner - Real-time planning system

Overview

RADIANT's orchestration system provides **17 reusable methods** that can be composed into **49 workflow patterns**. Each method is parameterized and can receive streams from previous methods in the pipeline.

Architecture



Stream Chaining

- Each method receives output from previous method(s) via `{{response}}` or `{{responses}}` template variables
- Methods can depend on multiple previous steps (`dependsOnSteps[]`)
- Parallel execution supported with `parallelExecution.enabled`

Output Stream Modes (NEW)

When a method uses N models, you can control how many streams come out:

Mode	Output	Description
`single`	1 stream	Synthesized result → `{{response}}` (default)
`all`	N streams	All model outputs → `{{responses}}` array
`top_n`	1-N streams	Best N by confidence → `{{responses}}` array
`threshold`	0-N streams	Only above confidence threshold → `{{responses}}` array

```
parallelExecution: {
  enabled: true,
  models: ['openai/o1', 'claude-3-5-sonnet', 'deepseek-reasoner'],
  outputMode: 'all',              // Pass all 3 streams to next step
  preserveModelAttribution: true  // Include model ID with each stream
}
```

Example: 3 models → 3 output streams

```
   Model 1            Model 2            Model 3
   (o1)              (Claude)          (DeepSeek)



            {{responses}} = [
              { modelId: 'o1', response: '...', confidence: 0.92 },
              { modelId: 'claude', response: '...', confidence: 0.88 },
              { modelId: 'deepseek', response: '...', confidence: 0.85 }
            ]



                Next Step
                SYNTHESIZE_RESPONSES
                receives 3 streams
```

Methods by Category

1. Generation Methods

GENERATE_RESPONSE **Purpose**: Generate a response to a prompt using specified model

Parameter	Type	Default	Description
`temperature`	number	0.7	Creativity/randomness (0-2)
`max_tokens`	integer	4096	Maximum output tokens

Prompt Template:

```
Generate a response to: {{prompt}}

Context: {{context}}
```

Recommended Models: Claude 3.5 Sonnet, GPT-4o, DeepSeek Chat

GENERATE_WITH_COT **Purpose**: Generate response using chain-of-thought reasoning

Parameter	Type	Default	Description
`temperature`	number	0.3	Lower for consistency
`max_tokens`	integer	8192	Extended for reasoning
`thinking_budget`	integer	2000	Tokens for reasoning

Prompt Template:

```
Think through this step-by-step before answering:

{{prompt}}

Show your reasoning, then provide your answer.
```

Recommended Models: OpenAI o1, DeepSeek Reasoner, Claude 3.5 Sonnet

---

REFINE_RESPONSE **Purpose**: Improve a response based on feedback

Parameter	Type	Default	Description
`refinement_focus`	string	"all"	Focus area: all, clarity, accuracy, completeness
`preserve_structure`	boolean	true	Keep original structure

Input Stream: `{{response}}` - Previous response to refine **Input Stream**: `{{feedback}}` - Critique or feedback

Prompt Template:

```
Improve this response based on the feedback:

Original Response: {{response}}

Feedback: {{feedback}}

Provide an improved response that addresses all feedback while maintaining the good parts.
```

---

2. Evaluation Methods

CRITIQUE_RESPONSE **Purpose**: Critically evaluate a response for flaws and improvements

Parameter	Type	Default	Description
`focus_areas`	array	[“accuracy”, “completeness”, “clarity”, “logic”]	What to evaluate
`severity_threshold`	string	“medium”	Minimum severity to report: low, medium, high

Input Stream: `{{response}}` - Response to critique

Prompt Template:

```
Critically evaluate this response:

Original Question: {{original_prompt}}
Response: {{response}}

Identify:
1. Factual errors
2. Logical flaws
3. Missing information
4. Clarity issues

For each issue, rate severity (low/medium/high) and suggest fixes.
```

Recommended Models: OpenAI o1, Claude 3.5 Sonnet

JUDGE_RESPONSES **Purpose**: Compare and judge multiple responses to select the best

Parameter	Type	Default	Description
`evaluation_mode`	enum	“pairwise”	pointwise, pairwise, listwise

Parameter	Type	Default	Description
`criteria`	array	[“accuracy”, “helpfulness”, “clarity”, “completeness”]	Evaluation criteria

Input Stream: `{{responses}}` - Array of responses to judge

Prompt Template:

```
Judge these responses to the question:

Question: {{original_prompt}}

{{#each responses}}Response {{@index}}: {{this}}

{{/each}}

Evaluate each on: {{criteria}}

Output: BEST: [number], SCORE: [0-1], REASONING: [explanation]
```

Output: `{ best: number, score: number, reasoning: string }`

VERIFY_FACTS **Purpose**: Extract and verify factual claims in a response

Parameter	Type	Default	Description
`extraction_method`	string	“explicit”	How to find claims: explicit, implicit, all
`verification_depth`	string	“thorough”	Verification level: quick, standard, thorough

Input Stream: `{{response}}` - Response to verify

Prompt Template:

```
Extract all factual claims from this response and verify each:

Response: {{response}}

For each claim:
1. State the claim
2. Verify if true/false/uncertain
3. Provide evidence or reasoning
4. Confidence level
```

GENERATE_CHALLENGE **Purpose**: Challenge a response by arguing the opposite position

Parameter	Type	Default	Description
`challenge_intensity`	string	"moderate"	How aggressive: mild, moderate, aggressive
`focus`	string	"weakest_points"	What to challenge: all, weakest_points, assumptions

Input Stream: `{{response}}` - Response to challenge

DEFEND_POSITION **Purpose**: Defend a response against challenges

Parameter	Type	Default	Description
`defense_strategy`	string	"address_all"	Strategy: address_all, prioritize, concede_gracefully
`concede_valid`	boolean	true	Acknowledge valid challenges

Input Streams: - `{{response}}` - Original response - `{{challenge}}` - Challenge to defend against

---

SELF_REFLECT **Purpose**: AI reflects on its own response to identify improvements

Parameter	Type	Default	Description
`reflection_depth`	string	"thorough"	Depth: quick, standard, thorough
`aspects`	array	["accuracy", "completeness", "clarity"]	What to reflect on

Input Stream: `{{response}}` - Response to reflect on

---

3. Synthesis Methods

SYNTHESIZE_RESPONSES **Purpose**: Combine best parts from multiple responses

Parameter	Type	Default	Description
`combination_strategy`	string	"best_parts"	Strategy: best_parts, merge, weighted

Parameter	Type	Default	Description
`conflict_resolution`	string	"majority"	How to resolve conflicts: majority, primary, newest

Input Stream: `{{responses}}` - Array of responses with model attribution

Prompt Template:

```
Synthesize these responses into one superior response:

Question: {{original_prompt}}

{{#each responses}}Response from {{model}}: {{content}}

{{/each}}

Create a response that:
1. Takes the best, most accurate parts from each
2. Resolves any conflicts
3. Is comprehensive and well-organized
```

BUILD_CONSENSUS **Purpose**: Identify points of agreement across multiple responses

Parameter	Type	Default	Description
`consensus_threshold`	number	0.7	Minimum agreement ratio (0-1)
`include_disputed`	boolean	true	Include disputed points with caveats

Input Stream: `{{responses}}` - Array of responses

---

4. Routing Methods

DETECT_TASK_TYPE **Purpose**: Analyze prompt to determine task type and complexity

Parameter	Type	Default	Description
`task_categories`	array	["coding", "reasoning", "creative", "factual", "math", "research"]	Categories to detect

Output:

```
{
  "taskType": "coding",
  "complexity": "complex",
  "requiredCapabilities": ["code_generation", "debugging"],
  "recommendedApproach": "chain_of_thought"
}
```

Recommended Models: GPT-4o-mini, Claude 3.5 Haiku (fast, cheap)

SELECT_BEST_MODEL **Purpose**: Choose the optimal model for a given task

Parameter	Type	Default	Description
`consider_cost`	boolean	true	Factor in cost
`consider_latency`	boolean	true	Factor in speed
`quality_priority`	number	0.7	Quality vs cost tradeoff (0-1)

Implementation: `code` - `model-selection-service.selectBestModel`

---

5. Reasoning Methods

DECOMPOSE_PROBLEM **Purpose**: Break down a complex problem into sub-problems

Parameter	Type	Default	Description
`max_subproblems`	integer	5	Maximum sub-problems
`decomposition_strategy`	string	"functional"	Strategy: functional, temporal, hierarchical

Prompt Template:

```
Decompose this problem into smaller sub-problems:

Problem: {{prompt}}

1. Identify independent components
2. Order by dependency
3. Estimate complexity of each
4. Return structured decomposition
```

Recommended Models: OpenAI o1, Claude 3.5 Sonnet

---

6. Aggregation Methods

MAJORITY_VOTE **Purpose**: Select the most common answer from multiple responses

Parameter	Type	Default	Description
`vote_method`	string	"exact_match"	Matching: exact_match, semantic, fuzzy
`tie_breaker`	string	"first"	Tie resolution: first, random, highest_confidence

Implementation: `code` - `aggregation-service.majorityVote`

WEIGHTED_AGGREGATE **Purpose**: Combine responses weighted by confidence/expertise

Parameter	Type	Default	Description
`weight_by`	string	"confidence"	Weight source: confidence, expertise, recency
`normalize`	boolean	true	Normalize weights to sum to 1

Implementation: `code` - `aggregation-service.weightedAggregate`

Workflow Patterns (49 Total)

Categories

Category	Count	Description
Adversarial & Validation	2	Security testing, vulnerability discovery
Debate & Deliberation	3	Multi-perspective analysis
Judge & Critic	3	Quality evaluation and improvement
Ensemble & Aggregation	3	Multi-model synthesis
Reflection & Self-Improvement	3	Iterative refinement
Verification & Fact-Checking	2	Accuracy validation
Multi-Agent Collaboration	2	Team-based problem solving
Reasoning Enhancement	9	CoT, ToT, ReAct, etc.
Model Routing Strategies	4	Optimal model selection
Domain-Specific Orchestration	4	Domain expertise routing
Cognitive Frameworks	14	First Principles, Systems Thinking, etc.

Pattern Quick Reference

Code	Name	Models	Latency	Quality Improvement
`CoT`	Chain-of-Thought	1	Medium	+20-40% on math/logic
`SCMR`	Majority Vote	3+	Medium	+15-25% accuracy
`ISFR`	Self-Refine Loop	1	High	+20-30% per iteration
`MDA`	Multi-Agent Debate	3+	Very High	+30-45% consensus
`CASCADE`	Cascade	2+	Variable	40-60% cost reduction
`ToT`	Tree-of-Thoughts	1	Very High	4%→74% on puzzles

Metrics Captured

For each method execution:

Metric	Description
`latencyMs`	Execution time in milliseconds
`costCents`	Cost in cents
`tokensUsed`	Input + output tokens
`modelUsed`	Model ID used
`qualityScore`	Auto-assessed quality (0-1)
`wasParallel`	Whether parallel execution was used
`parallelResults`	Individual model results if parallel
`iteration`	Iteration number for iterative methods

Aggregated Metrics (per workflow)

Metric	Description
`avgQualityScore`	Rolling average quality
`avgLatencyMs`	Average latency
`avgCostCents`	Average cost
`executionCount`	Total executions
`successRate`	Completion rate

Admin Configuration

Per-Tenant Customization

Admins can customize workflows:

```
{
  "workflowId": "uuid",
  "tenantId": "tenant-123",
  "configOverrides": {
    "temperature": 0.5,
    "max_iterations": 3
```

```
  },
  "disabledSteps": [3, 5],
  "modelPreferences": {
    "generator": "anthropic/claude-3-5-sonnet-20241022",
    "critic": "openai/o1"
  }
}
```

Parameter Schema

Each method has a JSON Schema for parameters:

```
{
  "type": "object",
  "properties": {
    "temperature": {
      "type": "number",
      "min": 0,
      "max": 2,
      "description": "Controls randomness"
    },
    "max_tokens": {
      "type": "integer",
      "description": "Maximum output length"
    }
  }
}
```

API Endpoints

Method Management

- `GET /api/admin/orchestration/methods` - List all methods
- `GET /api/admin/orchestration/methods/:code` - Get method details
- `PATCH /api/admin/orchestration/methods/:code` - Update method parameters

Workflow Management

- `GET /api/admin/orchestration/workflows` - List all workflows
- `GET /api/admin/orchestration/workflows/:code` - Get workflow with steps
- `POST /api/admin/orchestration/workflows/:code/customize` - Create tenant customization

Metrics

- `GET /api/admin/orchestration/metrics` - Aggregated metrics
- `GET /api/admin/orchestration/metrics/:workflowCode` - Workflow-specific metrics
- `GET /api/admin/orchestration/executions` - Recent executions

Stream Data Flow

Input Variables Available

Variable	Description	Source
`{{prompt}}`	Original user prompt	Request
`{{context}}`	Additional context	Request
`{{response}}`	Previous step output	Step N-1
`{{responses}}`	Multiple outputs	Parallel steps
`{{original_prompt}}`	Original prompt (unchanged)	Request
`{{feedback}}`	Critique output	Critic step
`{{challenge}}`	Challenge output	Challenger step

Output Structure

Each step produces:

```
{
  "response": "string",
  "tokens": 1234,
  "confidence": 0.85,
  "metadata": {}
}
```

For parallel execution with `outputMode: 'single'` (default):

```
{
  "response": "synthesized response string",
  "streamCount": 1,
  "outputMode": "single",
  "synthesisApplied": true,
  "modelsUsed": ["openai/o1", "claude-3-5-sonnet", "deepseek-reasoner"]
}
```

For parallel execution with `outputMode: 'all'`:

```
{
  "responses": [
    { "modelId": "openai/o1", "modelName": "o1", "response": "...", "confidence": 0.92, "laten
    { "modelId": "claude-3-5-sonnet", "modelName": "Claude 3.5 Sonnet", "response": "...", "co
    { "modelId": "deepseek-reasoner", "modelName": "DeepSeek Reasoner", "response": "...", "co
  ],
  "streamCount": 3,
  "outputMode": "all",
  "synthesisApplied": false,
  "modelsUsed": ["openai/o1", "claude-3-5-sonnet", "deepseek-reasoner"]
}
```

Multi-Model Parallel Execution

Overview

Any method can utilize **N models** simultaneously via the `parallelExecution` configuration. This enables: - **Consensus building** - Multiple perspectives on the same problem - **Quality improvement** - Best-of-N selection - **Robustness** - Fallback if one model fails - **Diverse outputs** - Different approaches to creative tasks

Parallel Execution Configuration

```
interface ParallelExecutionConfig {
  // Core settings
  enabled: boolean;
  mode: 'all' | 'race' | 'quorum';
  models: string[];

  // Synthesis
  synthesizeResults?: boolean;
  synthesisStrategy?: 'best_of' | 'merge' | 'vote' | 'weighted';
  weightByConfidence?: boolean;

  // AGI Dynamic Model Selection
  agiModelSelection?: boolean;
  minModels?: number;       // Default: 2
  maxModels?: number;       // Default: 5
  domainHints?: string[];

  // Failure handling
  timeoutMs?: number;
  quorumThreshold?: number; // For quorum mode: 0.5 = majority
  failureStrategy?: 'fail_fast' | 'continue' | 'fallback';

  // OUTPUT STREAM CONFIGURATION
  outputMode?: 'single' | 'all' | 'top_n' | 'threshold';
  outputTopN?: number;           // For top_n mode (default: 2)
  outputThreshold?: number;      // For threshold mode (default: 0.7)
  preserveModelAttribution?: boolean;
}
```

Execution Modes

Mode	Behavior	Use Case
`all`	Wait for all models to complete	Quality-critical tasks
`race`	Return first successful response	Latency-critical tasks
`quorum`	Wait for majority (threshold configurable)	Balanced approach

Synthesis Strategies

Strategy	Description
`best_of`	Select highest confidence response
`merge`	Combine all responses into one
`vote`	Majority answer wins
`weighted`	Weight by confidence scores

Output Stream Modes (Detailed)

Why Output Modes Matter

When a method uses 3 models, the question is: **how many streams should flow to the next step?**

- **Single stream** (default): Synthesize into one response for simple pipelines
- **All streams**: Pass all model outputs for the next step to compare/judge
- **Top N streams**: Only the best N by confidence
- **Threshold streams**: Only those above a quality bar

Mode Reference

`single` (Default)

```
3 Models    Synthesize    1 Stream    {{response}}
```

- **Use when**: Next step expects a single input
- **Output variable**: `{{response}}`
- **Example**: GENERATE → CRITIQUE (critic evaluates one response)

`all`

```
3 Models    3 Streams    {{responses}}[3]
```

- **Use when**: Next step needs to compare/synthesize multiple perspectives
- **Output variable**: `{{responses}}` array
- **Example**: 3x GENERATE → JUDGE_RESPONSES → pick best

`top_n`

```
3 Models    Sort by confidence    Top 2    {{responses}}[2]
```

- **Use when**: You want diversity but filtered by quality
- **Config**: `outputTopN: 2`
- **Output variable**: `{{responses}}` array

`threshold`

```
3 Models    Filter 80%    0-3 Streams    {{responses}}[0-3]
```

- **Use when**: Only high-quality responses should proceed
- **Config**: `outputThreshold: 0.8`
- **Output variable**: `{{responses}}` array (may be empty!)

Configuration Examples

Example 1: Multi-model critique with all perspectives

```
{
  stepName: 'Multi-Perspective Critique',
  method: 'CRITIQUE_RESPONSE',
  parallelExecution: {
    enabled: true,
    mode: 'all',
    models: ['openai/o1', 'claude-3-5-sonnet', 'deepseek-reasoner'],
    outputMode: 'all',                    // Pass all 3 critiques
    preserveModelAttribution: true        // Know which model said what
  }
}
// Next step receives {{responses}} with 3 critique objects
```

Example 2: Best-of-3 generation

```
{
  stepName: 'Generate with Best Selection',
  method: 'GENERATE_RESPONSE',
  parallelExecution: {
    enabled: true,
    mode: 'all',
    models: ['claude-3-5-sonnet', 'gpt-4o', 'gemini-pro'],
    outputMode: 'top_n',
    outputTopN: 1,                        // Only best response
    synthesizeResults: false              // Don't merge, just pick
  }
}
// Next step receives {{response}} (single best)
```

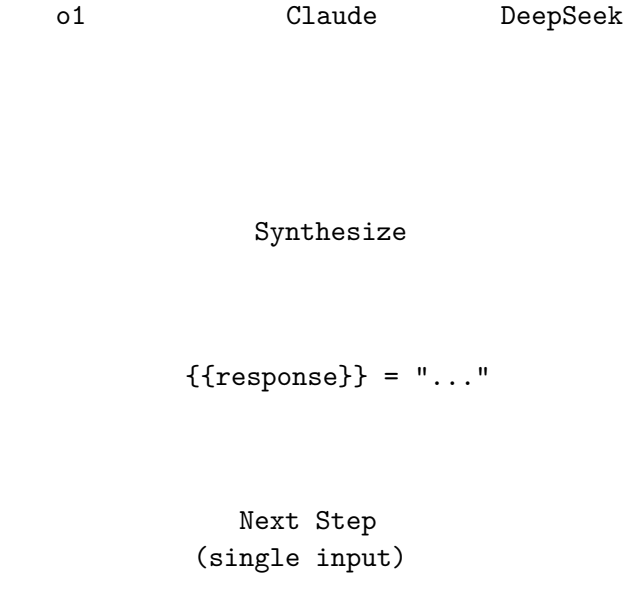
Example 3: Quality-filtered ensemble

```
{
  stepName: 'High-Confidence Ensemble',
  method: 'GENERATE_WITH_COT',
  parallelExecution: {
    enabled: true,
    agiModelSelection: true,              // AGI picks models
    minModels: 3,
    maxModels: 5,
    outputMode: 'threshold',
    outputThreshold: 0.85,                // Only 85% confidence
    preserveModelAttribution: true
```

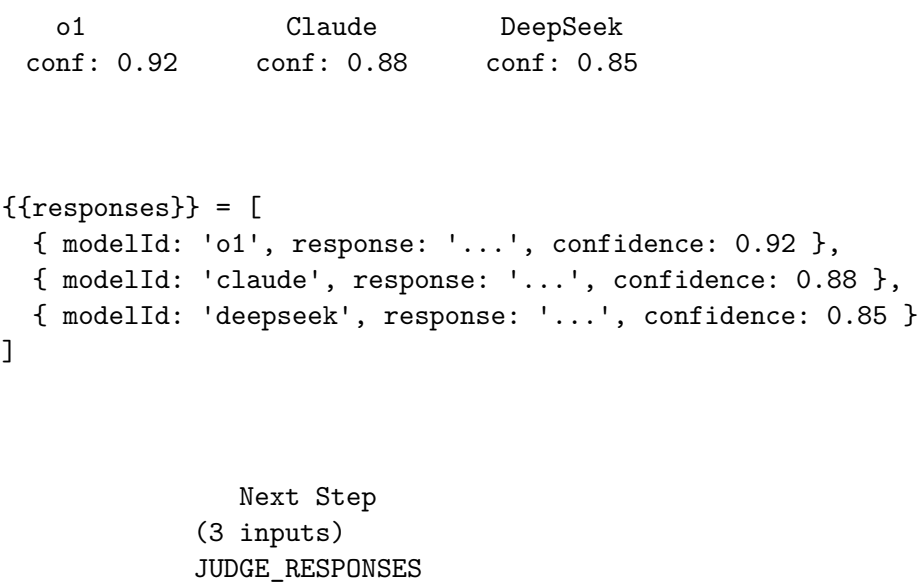
```
  }
}
// Next step receives {{responses}} with only high-confidence outputs
```

Stream Flow Diagrams

Single Mode (Default)



All Mode



Top N Mode (N=2)

```
   o1              Claude         DeepSeek
 conf: 0.92      conf: 0.88      conf: 0.85


                                        (filtered out)
{{responses}} = [
  { modelId: 'o1', response: '...', confidence: 0.92 },
  { modelId: 'claude', response: '...', confidence: 0.88 }
]


              Next Step
            (2 inputs)
```

Model Modes

Each model can run in a specialized mode for optimal performance:

Mode	Description	Best For
`standard`	Default execution	General tasks
`thinking`	Extended reasoning (o1, Claude thinking)	Complex logic
`deep_research`	In-depth research mode	Research tasks
`fast`	Speed-optimized (flash models)	Simple queries
`creative`	Higher temperature	Creative writing
`precise`	Low temperature, factual	Data extraction
`code`	Code-specialized	Programming
`vision`	Multimodal with vision	Image analysis
`long_context`	Extended context handling	Long documents

AGI Model Selection with Modes

When `agiModelSelection: true`, the system: 1. Analyzes the prompt/domain 2. Scores available models 3. Assigns optimal modes to each 4. Selects 2-5 models automatically

```
// AGI selection result
{
  selectedModels: [
    { modelId: 'openai/o1', mode: 'thinking' },
    { modelId: 'claude-3-5-sonnet', mode: 'standard' },
    { modelId: 'deepseek-reasoner', mode: 'deep_research' }
  ],
  reasoning: 'Selected 3 models with reasoning modes for complex analysis task',
  domainDetected: 'science',
```

```
  executionStrategy: 'parallel'
}
```

Proficiency System

Overview

The proficiency system is the **bridge between prompts and model selection**. It enables domain-aware orchestration by scoring both **domains** and **models** across 8 dimensions.

8 Proficiency Dimensions (1-10 scale)

Dimension	Description	High Score Means
`reasoning_depth`	Depth of logical reasoning required	Complex deduction, multi-step logic
`mathematical_quantitative`	Mathematical/quantitative analysis	Calculations, statistics, proofs
`code_generation`	Code writing/debugging capability	Programming tasks
`creative_generative`	Creative/generative content	Stories, art, brainstorming
`research_synthesis`	Research and synthesis ability	Literature review, analysis
`factual_recall_precision`	Factual accuracy requirements	Facts, definitions, dates
`multi_step_problem_solving`	Complex problem decomposition	Breaking down hard problems
`domain_terminology_handling`	Domain-specific jargon handling	Technical vocabulary

How Proficiencies Flow Through the System

```
USER PROMPT: "Derive the Navier-Stokes equations for incompressible flow"



STEP 1: DOMAIN DETECTION

Matched: Science → Physics → Fluid Dynamics
Confidence: 0.94

Detected Keywords: "Navier-Stokes", "equations", "incompressible", "derive"
```

```
STEP 2: PROFICIENCY EXTRACTION

Each level has proficiency scores that get MERGED:

Field (Science):              Domain (Physics):         Subspecialty (Fluid):
  reasoning: 8                  reasoning: 9              reasoning: 9
  math: 7                       math: 10                  math: 10
  code: 3                       code: 4                   code: 5
  creative: 4                   creative: 3               creative: 2
  research: 7                   research: 8               research: 7
  factual: 8                    factual: 9                factual: 8
  multi_step: 7                 multi_step: 9             multi_step: 10
  terminology: 6                terminology: 8            terminology: 9

MERGED PROFICIENCIES (weighted by specificity):
{
  reasoning_depth: 9,
  mathematical_quantitative: 10,
  code_generation: 5,
  creative_generative: 2,
  research_synthesis: 7,
  factual_recall_precision: 8,
  multi_step_problem_solving: 10,
  domain_terminology_handling: 9
}




STEP 3: ORCHESTRATION MODE SELECTION

Proficiency-based rules:
  reasoning_depth >= 9 AND multi_step >= 9 → extended_thinking
  code_generation >= 8 → coding
  creative_generative >= 8 → creative
  research_synthesis >= 8 → research
  mathematical_quantitative >= 8 → analysis

Selected: extended_thinking
Reason: "Complex reasoning required based on domain proficiencies"




STEP 4: MODEL MATCHING

Each model has proficiency scores. Match against domain requirements:
```

```
  Model: OpenAI o1                    Match Score: 94%
    reasoning: 10 (need 9)   +1
    math: 9 (need 10) -1
    multi_step: 10 (need 10)
    Strengths: [reasoning, multi_step, math]

  Model: Claude 3.5 Sonnet            Match Score: 87%
    reasoning: 9 (need 9)
    math: 8 (need 10) -2
    Strengths: [reasoning, research, terminology]

  Model: DeepSeek Reasoner            Match Score: 91%
    reasoning: 10 (need 9)   +1
    math: 10 (need 10)
    Strengths: [math, reasoning, code]

  SELECTED: o1 (primary), DeepSeek (fallback), Claude (fallback)




  STEP 5: EXECUTION

  parallelExecution: {
    enabled: true,
    models: ['openai/o1', 'deepseek-reasoner'],  // Both strong in math+reason
    mode: 'all',
    outputMode: 'top_n',
    outputTopN: 1,  // Pick best
    synthesisStrategy: 'best_of'
  }
```

Proficiency Types in the Hierarchy

```
Field (Top Level)
  field_proficiencies: ProficiencyScores

      Domain (Middle Level)
          domain_proficiencies: ProficiencyScores

              Subspecialty (Leaf Level)
                  subspecialty_proficiencies: ProficiencyScores
```

Model Proficiency Matching

```typescript
interface ModelProficiencyMatch {
  model_id: string;
  provider: string;
  model_name: string;
  match_score: number;           // 0-100 overall match
  dimension_scores: Record<ProficiencyDimension, number>;
  strengths: ProficiencyDimension[];
  weaknesses: ProficiencyDimension[];
  recommended: boolean;
  ranking: number;
}
```

Proficiency → Mode Decision Table

Proficiency Condition	Orchestration Mode	Reason
`reasoning_depth >= 9` AND `multi_step >= 9`	`extended_thinking`	Complex logical reasoning
`code_generation >= 8`	`coding`	Programming task
`creative_generative >= 8`	`creative`	Creative writing
`research_synthesis >= 8`	`research`	Research/analysis
`mathematical_quantitative >= 8`	`analysis`	Quantitative work
High `factual_recall` + sensitive topic	`self_consistency`	Accuracy critical
Default	`thinking`	Standard reasoning

Proficiency → Model Strengths Mapping

Model	Top Proficiencies	Best For
OpenAI o1	reasoning_depth (10), multi_step (10)	Complex reasoning
Claude 3.5 Sonnet	reasoning (9), research (9), terminology (9)	Research, analysis
DeepSeek Reasoner	math (10), reasoning (10), code (8)	Math, logic, code
GPT-4o	creative (8), research (8), factual (8)	General, creative
Gemini Pro	math (8), code (8), research (8)	Technical analysis
Claude Haiku	factual (7), terminology (7)	Quick answers

Example: Proficiency-Driven Workflow

```
// 1. Detect domain and get proficiencies
const detection = await domainTaxonomyService.detectDomain(prompt);
// Returns: { merged_proficiencies: { reasoning_depth: 9, math: 10, ... } }

// 2. Determine orchestration mode from proficiencies
const mode = determineOrchestrationMode(detection.merged_proficiencies);
// Returns: 'extended_thinking' (because reasoning >= 9 and multi_step >= 9)

// 3. Match models to proficiencies
const matches = await domainTaxonomyService.getMatchingModels(
  detection.merged_proficiencies,
  { max_models: 3, min_match_score: 80 }
);
// Returns: [{ model_id: 'o1', match_score: 94 }, { model_id: 'deepseek', match_score: 91 }]

// 4. Execute with matched models
const result = await orchestrationService.executeWorkflow({
  workflowCode: 'EXTENDED_THINKING_DUAL',
  parallelExecution: {
    enabled: true,
    models: matches.map(m => m.model_id),
    outputMode: 'top_n',
    outputTopN: 1
  }
});
```

Admin: Viewing Proficiencies

Admin Dashboard → Orchestration → Methods → Parallel & Streams tab

Shows: - Domain proficiency requirements for the task - Model match scores - Which dimensions drove model selection - Output stream configuration

---

AGI Brain + Workflow Integration

The AGI Brain Planner can **select and configure workflows** to solve problems. Users can either let the AGI choose the optimal workflow or specify their preferences.

User Choices for Workflows

```
interface GeneratePlanRequest {
  prompt: string;
  tenantId: string;
  userId: string;

  // ... other options ...
```

```
  // Workflow Selection - User choices
  preferredWorkflow?: string;           // User-selected workflow code
  workflowParameterOverrides?: Record<string, unknown>;  // User parameter tweaks
  allowAgiWorkflowSelection?: boolean;  // Let AGI pick workflow (default: true)
  excludeWorkflows?: string[];          // Workflows to exclude from selection
}
```

How AGI Selects Workflows

```
PROMPT: "Write a comprehensive analysis of renewable energy trends"



STEP 1: Check User Preference

Did user specify preferredWorkflow?
  YES → Use that workflow with user's parameter overrides
  NO  → Continue to AGI selection



STEP 2: AGI Workflow Selection

orchestrationPatternsService.selectPattern({
  prompt: "Write a comprehensive analysis...",
  taskType: "research",
  complexity: "complex",
  qualityPriority: 0.9,
  costSensitive: false,
  excludePatterns: user.excludeWorkflows
})

Scores 49 available workflows against problem characteristics



STEP 3: Selected Workflow

selectedWorkflow: {
  workflowCode: 'RESEARCH_SYNTHESIS_MULTI',
  workflowName: 'Multi-Model Research Synthesis',
  selectionReason: 'Matches research task, high quality priority',
  selectionConfidence: 0.89,
```

```
  selectionMethod: 'auto'
}

alternatives: [
  { workflowCode: 'CHAIN_OF_THOUGHT', matchScore: 0.82 },
  { workflowCode: 'SELF_CONSISTENCY', matchScore: 0.78 }
]




STEP 4: Workflow Steps with Parameters

workflowSteps: [
  {
    methodCode: 'DECOMPOSE_PROBLEM',
    parameterOverrides: { max_subproblems: 5 }
  },
  {
    methodCode: 'GENERATE_RESPONSE',
    isParallel: true,
    parallelConfig: {
      models: ['claude-3-5-sonnet', 'gpt-4o', 'gemini-pro'],
      outputMode: 'all'  // 3 streams to next step
    }
  },
  {
    methodCode: 'SYNTHESIZE_RESPONSES',
    parameterOverrides: { combination_strategy: 'best_parts' }
  }
]

workflowConfig: {
  ...defaultConfig,
  ...userParameterOverrides  // User's tweaks merged in
}
```

Example: User Specifies Workflow + Parameters

```
// User explicitly chooses a workflow and tweaks parameters
const plan = await agiBrainPlannerService.generatePlan({
  prompt: "Debug this React component that crashes on mount",
  tenantId: "tenant-123",
  userId: "user-456",

  // User choices
  preferredWorkflow: 'SELF_REFINE_LOOP',  // User picks this workflow
```

```
  workflowParameterOverrides: {
    max_iterations: 5,                    // More refinement rounds
    temperature: 0.2,                     // More precise
    refinement_focus: 'accuracy'
  }
});

// Result includes:
// - selectedWorkflow.selectionMethod = 'user'
// - workflowConfig merged with user overrides
// - workflowSteps configured with user parameters
```

Example: AGI Auto-Selects Workflow

```
// Let AGI choose the best workflow
const plan = await agiBrainPlannerService.generatePlan({
  prompt: "Compare the economic policies of three countries",
  tenantId: "tenant-123",
  userId: "user-456",

  allowAgiWorkflowSelection: true,  // default
  excludeWorkflows: ['FAST_SIMPLE']  // User says: not this one
});

// AGI analyzes prompt and selects:
// - selectedWorkflow.workflowCode = 'MULTI_AGENT_DEBATE'
// - selectedWorkflow.selectionMethod = 'domain_match'
// - selectedWorkflow.selectionReason = 'Comparison task benefits from debate'
// - alternatives = [other matching workflows]
```

Plan Output with Workflow

```
interface AGIBrainPlan {
  // ... existing fields ...

  // Workflow Integration
  selectedWorkflow?: {
    workflowId: string;
    workflowCode: string;
    workflowName: string;
    description: string;
    category: string;
    selectionReason: string;
    selectionConfidence: number;
    selectionMethod: 'auto' | 'user' | 'domain_match';
  };

  workflowSteps?: Array<{
```

```
    bindingId: string;
    stepOrder: number;
    methodCode: string;
    methodName: string;
    parameterOverrides: Record<string, unknown>;
    dependsOn: string[];
    isParallel: boolean;
    parallelConfig?: {
      models: string[];
      outputMode: 'single' | 'all' | 'top_n' | 'threshold';
    };
  }>;

  workflowConfig?: Record<string, unknown>;

  alternativeWorkflows?: Array<{
    workflowCode: string;
    workflowName: string;
    matchScore: number;
    reason: string;
  }>;
}
```

Specialty Categories (Domain Expertise)

> **Full Documentation**: See SPECIALTY-RANKING.md for complete details on the specialty ranking system, AI-powered research, admin controls, and database schema.

In addition to the 8 proficiency dimensions, models are ranked across **20 specialty categories** representing domain-specific expertise:

Specialty Categories

Category	Icon	Description
`reasoning`		Reasoning & Logic
`coding`		Code Generation
`math`		Mathematics
`creative`		Creative Writing
`analysis`		Data Analysis
`research`		Research & Synthesis
`legal`		Legal & Compliance
`medical`		Medical & Healthcare
`finance`		Finance & Trading
`science`		Scientific
`debugging`		Debugging & QA
`architecture`		System Architecture

Category	Icon	Description
security		Security
vision		Vision & Images
audio		Audio & Speech
conversation		Conversational
instruction		Instruction Following
speed		Low Latency
accuracy		High Accuracy
safety		Safety & Alignment

Two-Layer Proficiency System

```
LAYER 1: TASK PROFICIENCY DIMENSIONS (8)
From domain taxonomy - "What capabilities does this task need?"

  reasoning_depth              Multi-step logical thinking
  mathematical_quantitative    Calculations, proofs, statistics
  code_generation              Programming tasks
  creative_generative          Stories, art, ideas
  research_synthesis           Literature review, analysis
  factual_recall_precision     Facts, definitions, accuracy
  multi_step_problem_solving   Breaking down complex problems
  domain_terminology_handling  Technical vocabulary


                                  Drives


LAYER 2: SPECIALTY CATEGORIES (20)
Per-model rankings - "How good is each model in each specialty?"

Domain Expertise:                 Performance Attributes:
    medical                         speed
    legal                           accuracy
    finance                         safety
    science                         instruction
    security
    architecture                Modalities:
                                       vision
Task Capabilities:                     audio
    reasoning
    coding
    math
    creative
    analysis
    research
```

```
    debugging
    conversation
```

How Both Layers Work Together

```
PROMPT: "Analyze this ECG reading and suggest treatment options"



STEP 1: Domain Detection
→ Field: Medicine → Domain: Cardiology → Subspecialty: Diagnostics
→ Confidence: 0.91



STEP 2: Extract TASK PROFICIENCY Requirements

From domain taxonomy:
{
  reasoning_depth: 8,              // Diagnostic reasoning
  mathematical_quantitative: 5,    // Some measurements
  factual_recall_precision: 9,     // Medical accuracy critical
  research_synthesis: 7,           // Treatment guidelines
  domain_terminology_handling: 10  // Medical jargon
}



STEP 3: Query SPECIALTY RANKINGS for Models

Required specialties: medical + accuracy + safety + research

Model Specialty Scores:

 Model                    Medical   Accuracy   Safety   Research

 Claude 3.5 Sonnet      92 (A)    91 (A)    95 (S)   90 (A)
 GPT-4o                 88 (A)    89 (A)    90 (A)   87 (A)
 DeepSeek Medical*      95 (S)    85 (B)    88 (A)   82 (B)
 Gemini Pro             84 (B)    86 (B)    89 (A)   88 (A)


* Self-hosted domain-specific model
```

```
STEP 4: Combined Scoring

Final Score = TaskProficiencyMatch × SpecialtyScore × SafetyWeight

Claude 3.5 Sonnet: 0.88 × 92 × 1.2 = 97.2 ← SELECTED (primary)
DeepSeek Medical:  0.82 × 95 × 1.0 = 77.9 ← SELECTED (fallback)
GPT-4o:            0.85 × 88 × 1.1 = 82.3 ← SELECTED (fallback)



STEP 5: Execution with Multi-Model

parallelExecution: {
  enabled: true,
  models: ['claude-3-5-sonnet', 'deepseek-medical', 'gpt-4o'],
  outputMode: 'threshold',
  outputThreshold: 0.85,  // Only high-confidence medical advice
  synthesisStrategy: 'weighted'  // Weight by specialty scores
}
```

Specialty Ranking Structure

```typescript
interface SpecialtyRanking {
  rankingId: string;
  modelId: string;
  provider: string;
  specialty: SpecialtyCategory;     // 'medical', 'legal', 'coding', etc.
  proficiencyScore: number;          // 0-100 overall score
  benchmarkScore: number;            // 0-100 from published benchmarks
  communityScore: number;            // 0-100 from community reviews
  internalScore: number;             // 0-100 from internal usage data
  rank: number;                      // Global rank for this specialty
  percentile: number;                // e.g., top 10%
  tier: 'S' | 'A' | 'B' | 'C' | 'D' | 'F';  // Quality tier
  confidence: number;                // 0-1 confidence in assessment
  trend: 'improving' | 'stable' | 'declining';
  adminOverride?: number;            // Admin can lock a score
  isLocked: boolean;
}
```

Tier System

Tier	Score Range	Description
S	95-100	Elite - Best-in-class for this specialty
A	85-94	Excellent - Highly recommended
B	75-84	Good - Solid performance
C	65-74	Average - Acceptable
D	50-64	Below Average - Use with caution
F	0-49	Poor - Not recommended

Example: Model Specialty Profiles

Claude 3.5 Sonnet

```
reasoning:      94 (S)
coding:         95 (S)
math:           88 (A)
creative:       92 (A)
medical:        92 (A)
legal:          89 (A)
security:       91 (A)
safety:         95 (S)
```

OpenAI o1

```
reasoning:      98 (S)
coding:         90 (A)
math:           96 (S)
creative:       75 (B)
medical:        85 (B)
legal:          88 (A)
security:       89 (A)
safety:         92 (A)
```

DeepSeek Coder

```
reasoning:      85 (B)
coding:         96 (S)
math:           92 (A)
creative:       65 (C)
debugging:      94 (S)
architecture: 88 (A)
speed:          90 (A)
```

AI-Powered Research

The specialty rankings are maintained through **automated AI research**:

```
// Research model proficiency across all specialties
const result = await specialtyRankingService.researchModelProficiency('anthropic/claude-3-5-so
```

```
// Research all models for a specific specialty
const result = await specialtyRankingService.researchSpecialtyRankings('medical');
```

Research sources include: - Published benchmarks (MMLU, HumanEval, MATH, etc.) - Community reviews and feedback - Internal usage data and quality scores - Domain-specific evaluations

Admin Controls

Admins can: - **Override scores**: Lock a model's specialty score - **View leaderboards**: See top models per specialty - **Trigger research**: Refresh rankings from latest data - **Configure weights**: Adjust benchmark vs community vs internal weighting