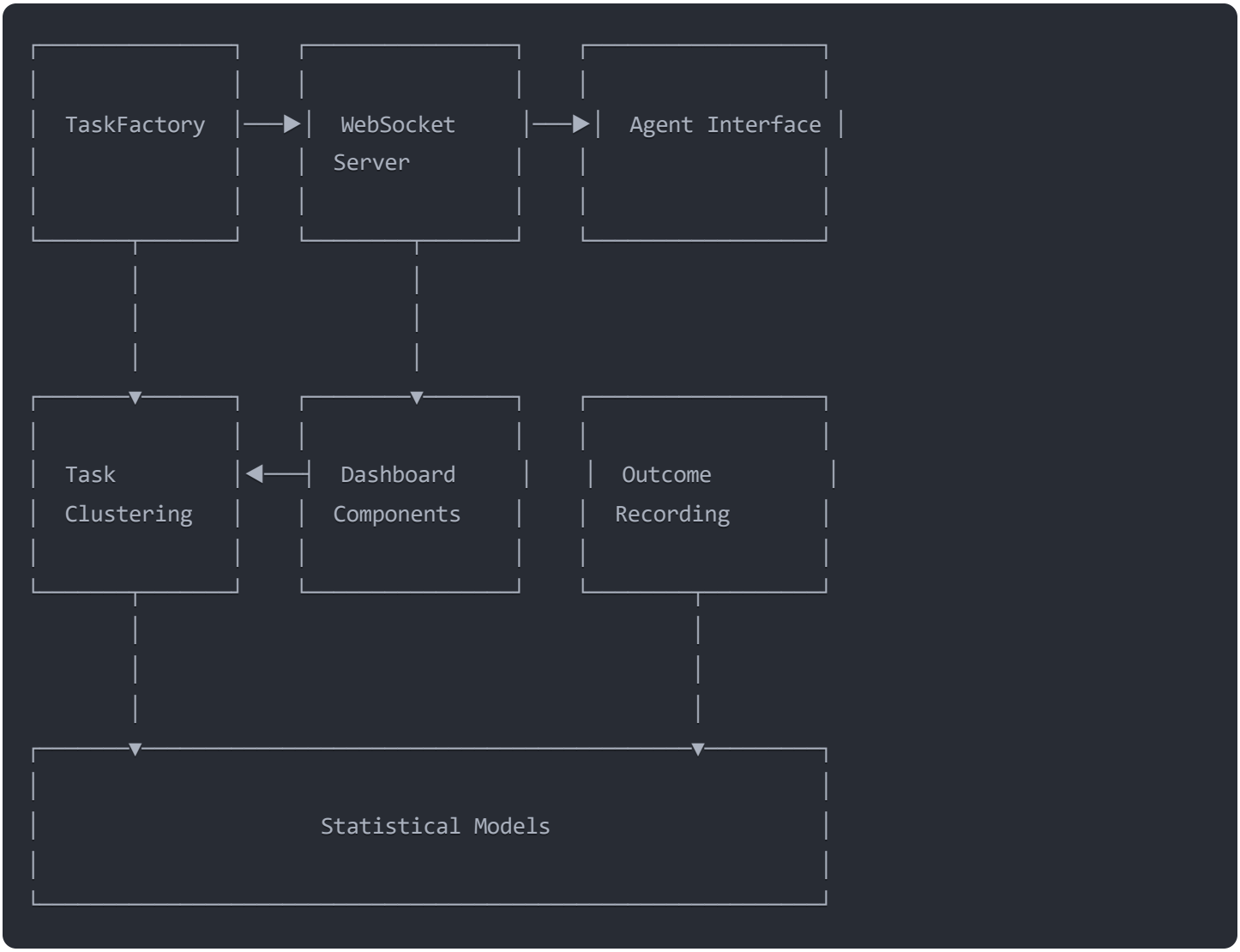
Shared Knowledge Base: TaskFactory System

System Architecture

Core Components

Copy



Data Flow

1. **Task Creation & Classification**
   - Task request comes through WebSocket server
   - TaskFactory analyzes content and assigns reasoning effort
   - Task dispatched to agent with reasoning strategy
2. **Task Execution & Monitoring**
   - Agent processes task according to reasoning strategy
   - Dashboard displays real-time status and complexity metrics

- Task clustering system builds typology in background

3. **Outcome Recording & Learning**
   - Task results captured with performance metrics
   - Statistical models update based on actual outcomes
   - System adjusts weights and thresholds automatically

Common Data Models

Core Models

python Copy

```python
class ReasoningEffort(str, Enum):
    LOW = "low"
    MEDIUM = "medium"
    HIGH = "high"

class ReasoningStrategy(str, Enum):
    DIRECT = "direct_answer"
    COT = "chain-of-thought"
    COD = "chain-of-draft"
```

Task Model

python Copy

```python
class Task(BaseModel):
    task_id: str
    agent: str  # Originating agent/user
    content: str
    target_agent: str
    event: TaskEvent
    intent: MessageIntent
    confidence: Optional[float]
    reasoning_effort: ReasoningEffort
    reasoning_strategy: ReasoningStrategy
    metadata: Optional[Dict[str, Any]]
```

TaskFactory Output Format

python Copy

```
# Return format of estimate_reasoning_effort function
{
    "word_count": 15,
    "complexity_score": 3.5,
    "category_scores": {
        "analytical": 1,
        "comparative": 0,
        "creative": 1,
        "complex": 1
    },
    "matched_keywords": {
        "analytical": ["analyze"],
        "creative": ["design"],
        "complex": ["simulate"]
    },
    "base_effort": "medium",
    "thresholds": {
        "high": 35.0,
        "medium": 16.0
    },
    "event_adjustment": "Increased to HIGH due to refine event",
    "final_effort": "high"
}
```

Integration Points

WebSocket Message Format

json

Copy

```
{
  "type": "task",
  "payload": {
    "task_id": "task_abc123",
    "content": "Analyze and optimize this algorithm",
    "agent": "user",
    "target_agent": "claude",
    "event": "plan",
    "intent": "start_task",
    "reasoning_effort": "high",
    "reasoning_strategy": "chain-of-draft"
  },
  "timestamp": "2023-12-15T14:30:45.123Z"
}
```

Task Outcome Recording

python

Copy

```
# Call format
TaskFactory.record_task_outcome(
    task_id="task_abc123",
    diagnostics=task_diagnostics,  # Original diagnostics from task creation
    actual_duration=182.5,         # Seconds to complete
    success=True,                  # Whether task was successful
    feedback="Task complexity was appropriate"  # Optional
)
```

Clustering System Output

json

Copy


```json
{
  "clusters_found": 5,
  "cluster_profiles": {
    "0": {
      "size": 45,
      "avg_complexity": 3.2,
      "avg_word_count": 18.5,
      "dominant_effort": "high",
      "effort_distribution": {"high": 0.65, "medium": 0.35, "low": 0.0},
      "success_rate": 0.78,
      "examples": ["Analyze this dataset...", "Design an algorithm..."]
    }
  },
  "best_agents_by_cluster": {
    "0": "claude",
    "1": "gpt",
    "2": "grok"
  }
}
```


Technical Environment

Redis Configuration

- **Host**: `redis` (when running inside Docker)
- **Port**: `6379`
- **Channels**:
  - `task_queue`: Task assignment queue
  - `responses_channel`: Agent responses
  - `task_events`: Task status updates

Required Environment Variables

Copy

```
# TaskFactory Configuration
TASK_FACTORY_AUTOTUNE=true
TASK_FACTORY_RETAIN_HISTORY=true
TASK_FACTORY_HISTORY_LIMIT=1000
TASK_FACTORY_MIN_SAMPLES=10

# Agent Configuration
REQUIRED_AGENTS=Grok,Claude,GPT,Gemini
DEFAULT_AGENT=Grok

# Redis Configuration
REDIS_HOST=redis
REDIS_PORT=6379
```

Resource Requirements

- Python 3.10+
- Node.js 16+ (for frontend)
- 8GB+ RAM
- SentenceTransformer models
- Redis 6+

Test Datasets

A synthetic test dataset is provided with:

- 200 sample tasks
- Varying complexity levels
- Predefined categories
- Expected reasoning efforts
- Sample agent outcomes

Location: `/shared/test_data/synthetic_tasks.json`

Best Practices

1. **Decoupled Components**: Use clearly defined interfaces
2. **Event-driven Communication**: Rely on Redis pubsub

3. **Comprehensive Logging**: Log all decision factors
4. **Graceful Degradation**: Default to MEDIUM effort when in doubt
5. **Continuous Integration**: Use the shared test datasets

Version Control

- Branch naming: `component/feature-name`
- Commit messages: `[Component] Brief description of change`
- Pull request template available at `.github/PULL_REQUEST_TEMPLATE.md`

Support & Troubleshooting

- Shared Slack channel: #taskfactory-integration
- Knowledge base: https://confluence.internal/TaskFactory
- Log central: Grafana dashboard at https://logs.internal/taskfactory

Let's kick some technological ass! 🚀