LLM Task Force Assignment Briefs

Overview

We're building an advanced multi-agent reasoning system with dynamic effort assessment, agent specialization, and intelligent task routing. Each LLM will focus on specific components based on their strengths, with clear integration points to ensure the system works together seamlessly.

Shared Resources

All LLMs will have access to:

- System architecture diagrams
- Data models and schemas (TaskFactory, TaskRouter, etc.)
- Integration contract specifications
- Common test datasets
- Redis and WebSocket configurations

Claude Assignment

Core Focus: TaskFactory implementation and evolution roadmap

Primary Deliverables

1. **Enhanced TaskFactory Module**
   - Dynamic reasoning effort assessment (LOW/MEDIUM/HIGH)
   - Weighted keyword categories with automatic adjustment
   - Category impact analysis and auto-tuning mechanism
   - Diagnostic output and metadata generation
2. **Integration Guide Documentation**
   - Comprehensive integration steps for all components
   - Best practices for agent adaptation
   - Performance optimization guidelines
   - Testing and validation frameworks
3. **Evolution Roadmap**
   - Path from rule-based to statistical learning
   - Transition plan to neural router architecture

- Metrics for success measurement
- Future enhancement possibilities

Interface Contracts

- `estimate_reasoning_effort(content, event, intent, confidence)` → (ReasoningEffort, diagnostics)
- `record_task_outcome(task_id, diagnostics, duration, success, feedback)` → None
- `create_task(agent, content, target_agent, ...)` → (Task, diagnostics)

Performance Requirements

- Reasoning effort assessment accuracy >90%
- TaskFactory overhead <50ms per task
- Auto-tuning after 100+ task outcomes
- Comprehensive diagnostics for every task

GPT Assignment

Core Focus: WebSocket server integration and frontend dashboard

Primary Deliverables

1. **WebSocket Server Implementation**
   - TaskFactory integration for all incoming tasks
   - Reasoning strategy transmission to agents
   - Task outcome recording mechanism
   - Real-time status updates for clients
2. **Frontend Dashboard Components**
   - Task complexity visualization dashboard
   - Effort distribution breakdown views
   - Agent performance metrics display
   - Task detail inspection interface
3. **API Endpoint Design**
   - RESTful endpoints for task management
   - Analytics data retrieval endpoints
   - WebSocket event protocol specification

- Authentication and rate limiting

Interface Contracts

- WebSocket event format: `{type, payload, timestamp, source}`
- RESTful endpoints: `GET /tasks`, `POST /tasks`, `GET /analytics`
- Dashboard API contract: `fetchTaskData()`, `fetchAnalytics()`

Performance Requirements

- WebSocket server handles 100+ concurrent connections
- Dashboard loads in <2s with 1000+ tasks
- Real-time updates with <100ms latency
- Graceful degradation under high load

Grok Assignment

Core Focus: Task Clustering System and Router optimization

Primary Deliverables

1. **Task Clustering System**
   - Content embedding and feature extraction
   - Cluster identification algorithm
   - Visualization tools for cluster analysis
   - Agent-to-cluster performance tracking
2. **Router Optimization Algorithms**
   - Multi-factor routing logic
   - Exploration/exploitation balancing
   - Performance feedback integration
   - Agent specialization reinforcement
3. **Novel Routing Patterns**
   - Deadline-aware routing strategies
   - Confidence-based load balancing
   - Error recovery routing patterns
   - A/B testing framework for routing strategies

Interface Contracts

- `cluster_tasks(task_history)` → cluster_analysis
- `predict_cluster(content, complexity, categories)` → cluster_id
- `route_task(task_id, content, available_agents, diagnostics)` → (agent, routing_metadata)

Performance Requirements

- Clustering analysis completes in <10s for 1000 tasks
- Routing decision overhead <10ms per task
- Cluster prediction accuracy >80%
- Router improves agent success rate by >15%

Gemini Assignment

Core Focus: Data pipeline and statistical model development

Primary Deliverables

1. **Data Collection Pipeline**
   - Standardized outcome recording service
   - Task history database schema
   - Aggregation jobs for analytics
   - Data validation and cleaning tools
2. **Statistical Models for Effort Prediction**
   - Feature engineering pipeline
   - Ensemble model for effort prediction
   - Confidence scoring mechanism
   - A/B testing framework
3. **Visualization Components**
   - UMAP-based cluster visualization
   - Performance metrics dashboards
   - Learning curve and improvement tracking
   - Interactive exploration tools

Interface Contracts

- `train_effort_predictor(training_data)` → model
- `predict_effort(content, context_features)` → (effort, confidence)
- `generate_visualizations(cluster_data)` → visualization_paths

Performance Requirements

- Prediction model >85% accuracy
- Training pipeline runs in <30 minutes
- Visualization generation <5s per view
- Performance improvement tracking with <2% error margin

Integration Checkpoints

Week 1: Foundation Integration

- Basic TaskFactory + WebSocket server
- Simple dashboard visualization
- Initial data recording pipeline

Week 2: Advanced Integration

- Clustering system + TaskFactory
- Router + WebSocket server
- Statistical models + data pipeline

Week 3: Full System Integration

- All components working together
- End-to-end testing with synthetic tasks
- Performance benchmarking and optimization

Success Criteria

1. **System Performance**
   - Task success rate improves by >20%
   - Agent utilization balance improves by >30%
   - Overall system throughput increases by >25%
2. **User Experience**
   - Dashboard provides clear insights into task complexity

- Task allocation "feels right" to human reviewers
- System becomes more accurate over time

3. **Technical Achievement**
   - Auto-tuning system successfully adjusts weights
   - Cluster-based routing properly specializes agents
   - Statistical models outperform rule-based assessment

Next Steps

Once all LLMs have reviewed and accepted their assignments, we'll:

1. Set up the shared knowledge repository
2. Establish integration test framework
3. Create the CI/CD pipeline
4. Begin parallel development

Let's kick some technological ass together! 🚀