Praval v0.7.6: Multi-Agent AI Framework

Simple Agents, Powerful Results

Rajesh S (@aiexplorations)

October 2025

Figure 1: Praval Logo

Praval (pravālam - Sanskrit for "coral") - a multi-agent AI framework that enables the creation of complex, interconnected agent systems from simple components.

The Praval Philosophy

The Coral Reef Metaphor

Just as coral reefs create complex, thriving ecosystems from countless simple organisms working together, Praval enables sophisticated AI systems through the collaboration of specialized agents. Each agent excels at one thing, but together they achieve intelligence beyond what any individual could accomplish.

Core Principles

- **Simplicity Over Complexity**
  - Intelligence emerges from simple, specialized agents working together
  - No "god agents" that try to do everything
  - Each agent has a clear, focused purpose
- **Collaboration Over Control**
  - Agents communicate naturally through message-passing
  - No centralized orchestration required
  - Systems self-organize through agent interactions
- **Identity Over Instructions**

- Define agents by *what they are*, not *what they do*
- Behavioral consistency through identity-driven design
- Natural adaptation to new situations within their domain

- **Emergence Over Engineering**
  - Complex behaviors arise from simple interactions
  - Systems evolve and improve over time
  - New capabilities emerge without rewrites

What's Possible with Praval

Core Capabilities Checklist

Agent Creation & Management

- ☒ Create specialized agents with clear identities
- ☒ Deploy agents using simple Python decorators
- ☒ Register agents for discovery and coordination
- ☒ Monitor agent performance and health

Communication & Collaboration

- ☒ Natural message-passing between agents
- ☒ Channel-based communication for organization
- ☒ Broadcast messages to multiple agents
- ☒ Event-driven agent responses

Memory & Learning

- ☒ Persistent memory across sessions
- ☒ Contextual conversation history
- ☒ Knowledge accumulation over time
- ☒ Semantic search across memories

System Orchestration

- ☒ Multi-agent workflows and pipelines
- ☒ Self-organizing agent networks
- ☒ Fault-tolerant system behavior
- ☒ Dynamic agent discovery

Production Features

- ☒ Multiple LLM provider support (OpenAI, Anthropic, Cohere)
- ☒ Docker deployment with full stack
- ☒ Comprehensive testing framework
- ☒ Performance monitoring and optimization
- ☒ Enterprise-grade security (Secure Spores)
- ☒ Multi-protocol messaging (AMQP, MQTT, STOMP)
- ☒ Production-ready memory system with ChromaDB
- ☒ Tool system with @tool decorator for external integrations
- ☒ PDF processing and knowledge base indexing
- ☒ Collection separation architecture (knowledge vs conversational memory)
- ☒ Unified data storage system (PostgreSQL, Redis, S3, MinIO)

Use Case Examples

Knowledge Processing Systems

- Document analysis with specialized extractors and analyzers
- Research assistance with domain experts and synthesizers
- Content generation with writers, editors, and reviewers

Business Intelligence

- Market analysis with data collectors and trend analyzers
- Customer support with specialists for different issue types
- Process automation with workflow-specific agents

Creative Applications

- Collaborative writing with idea generators and editors
- Problem-solving with diverse thinking styles
- Innovation labs with complementary specialist perspectives

Educational Systems

- Personalized tutoring with subject matter experts
- Interactive learning with question generators and explainers
- Adaptive curriculum with progress trackers and content creators

---

Getting Started

1. Simple First Agent

```
from praval import agent, chat

@agent("helpful_assistant")
def my_first_agent(spore):
    user_message = spore.knowledge.get("message")
    response = chat(f"Help the user with: {user_message}")
    return {"response": response}
```

2. Agent Collaboration

```
@agent("question_generator")
def ask_questions(spore):
    topic = spore.knowledge.get("topic")
    question = chat(f"Ask an interesting question about {topic}")
    broadcast({"type": "question_ready", "question": question})

@agent("answer_provider", responds_to=["question_ready"])
def provide_answers(spore):
    question = spore.knowledge.get("question")
    answer = chat(f"Provide a thorough answer: {question}")
    return {"answer": answer}
```

3. Memory-Enabled Learning

```
@agent("learning_assistant", memory=True)
def smart_assistant(spore):
    # Remembers previous conversations
    # Learns from each interaction
    # Provides personalized responses
    return process_with_memory(spore)
```

4. Tool-Enabled Agents (v0.7.3)

```python
from praval import agent, tool

# Define reusable tools
@tool("web_search", category="research")
def search_web(query: str) -> dict:
    """Search the web for information."""
    return {"results": perform_search(query)}

@tool("calculate", category="math")
def calculator(expression: str) -> float:
    """Evaluate mathematical expressions."""
    return eval(expression)

# Agent with automatic tool access
@agent("research_assistant")
def researcher(spore):
    # Tools are automatically available
    query = spore.knowledge.get("query")
    results = search_web(query)
    return {"findings": results}
```

5. Knowledge Base with PDF Support (v0.7.0)

```python
from praval import agent

@agent("document_analyst",
       memory=True,
       knowledge_base="./documents/")  # Auto-indexes PDFs
def analyze_docs(spore):
    """I analyze documents and answer questions about them."""
    question = spore.knowledge.get("question")

    # Automatically searches indexed PDFs in knowledge base
    relevant_info = search_knowledge_base(question)
    answer = chat(f"Answer based on: {relevant_info}\nQ: {question}")

    return {"answer": answer}
```

6. Secure Enterprise Communication (v0.6.0)

```python
from praval.core.secure_spore import SecureSporeFactory, SporeKeyManager

# Initialize secure messaging
key_manager = SporeKeyManager("secure_agent")
secure_factory = SecureSporeFactory(key_manager)

# End-to-end encrypted communication
secure_spore = secure_factory.create_secure_spore(
    to_agent="target_agent",
    knowledge={"classified": "information"},
    recipient_public_keys=recipient_keys
)
```

Version 0.7.x Evolution

v0.7.6: Collection Separation Architecture (Latest)

Intelligent Memory Organization: - **Separate Collections**: Knowledge base and conversational memory now use distinct ChromaDB collections - **Immutable Knowledge**: Knowledge base content is protected - semantic memories cannot be deleted - **Smart Routing**: Automatic routing based on memory type (semantic → knowledge, episodic/working → memory) - **Cross-Collection Operations**: Search, retrieve, and analyze across both collections seamlessly

Data Integrity & Security: - Selective deletion policy - only conversational memory can be cleared - Knowledge base preservation ensures important information is never lost - Automatic migration from legacy single-collection setups with zero downtime - Enhanced statistics showing separate metrics for knowledge vs conversations

v0.7.0-0.7.5: Knowledge Base & Tool System

PDF Processing (v0.7.0): - Native PDF text extraction using PyPDF2 - Intelligent text cleaning (URLs, emails, formatting) - Automatic PDF indexing into knowledge base - Support for multi-page document analysis

Tool System (v0.7.3): - **@tool Decorator**: Declarative tool definition for external integrations - **ToolRegistry**: Centralized tool management and discovery - **Agent Integration**: Tools automatically exposed through enhanced @agent decorator - **Categorization**: Organize tools by function, domain, or capability

Robustness Improvements (v0.7.1, v0.7.4, v0.7.5): - Fixed ChromaDB search query syntax for complex conditions - Automatic collection creation with proper error handling - Knowledge base benchmarking and performance validation - Enhanced pytest markers for organized testing (unit, integration, performance)

v0.6.1: Unified Data Storage System

Enterprise Storage Integration: - **Storage Providers**: PostgreSQL, Redis, S3/MinIO, Qdrant, FileSystem - **Storage Decorators**: `@storage_enabled()` and `@requires_storage()` for declarative access - **Data References**: Lightweight sharing of large datasets through spore communication - **Cross-Storage Operations**: Query and manage data across multiple backends simultaneously

Production Features: - Async connection pooling and health monitoring - Permission-based access control per agent - Smart storage selection based on data characteristics - Batch operations for high-throughput scenarios

v0.6.0: Secure Spores & Multi-Agent Fix

Critical Bug Fix: Multi-agent communication now works correctly with proper channel subscription

Military-Grade Security: - End-to-end encryption (Curve25519 + XSalsa20 + Poly1305) - Digital signatures with Ed25519 for message authenticity - Perfect forward secrecy and automatic key rotation

Multi-Protocol Transport: - AMQP for enterprise message brokers (RabbitMQ, Azure Service Bus) - MQTT for IoT and edge computing - STOMP for enterprise integration - Unified abstraction across all protocols

---

The Praval Advantage

Maintainability

- Small, focused agents are easy to understand and modify
- Clear separation of concerns prevents cascading changes
- Modular architecture enables independent development

Testability

- Individual agents can be tested in isolation
- Predictable behavior through identity-driven design
- Comprehensive test coverage for production confidence

Scalability

- Add new capabilities by creating new agents
- Linear complexity growth instead of exponential
- Natural load distribution across specialists

Robustness

- Graceful degradation when agents fail
- Fault isolation prevents system-wide crashes
- Self-healing through agent redundancy

Innovation

- Rapid prototyping of new capabilities
- Easy experimentation with different agent combinations
- Natural evolution of system intelligence

The Vision: Collaborative Intelligence

Praval represents a fundamental shift from building monolithic AI systems to creating **ecosystems of collaborative intelligence**.

Where We're Heading

- **Understandable AI**: Systems built from comprehensible, specialized parts
- **Adaptive Intelligence**: Systems that learn and evolve without complete rewrites
- **Natural Behavior**: AI that emerges from collaboration rather than programming
- **Sustainable Development**: Growth through addition, not reconstruction

The Developer Journey

1. **Start Simple**: Begin with basic agent interactions and clear communication
2. **Embrace Specialists**: Create focused agents instead of generalist solutions
3. **Watch Emergence**: Let intelligent behaviors arise from agent collaboration
4. **Scale Naturally**: Add capabilities by introducing new specialist agents

---

Quick Start

Installation

```
# Basic installation
pip install -r requirements.txt

# Development installation (for contributors)
pip install -e .  # Editable install - changes take effect immediately

# Docker deployment (recommended for production)
```

```
docker-compose up -d  # Basic stack with ChromaDB
docker-compose -f docker-compose.secure.yml up -d  # Secure enterprise stack

# Containerized examples (v0.6.2+)
cd examples/docker-examples/005-memory-agents
./run-memory-demo.sh  # Memory-enabled agent demonstration

cd examples/docker-examples/010-unified-storage
./run-storage-demo.sh  # Full-stack demo with PostgreSQL, Redis, MinIO
```

First Steps

1. **Run the Examples**:
   - Start with `examples/pythonic_knowledge_graph.py` for core concepts
   - Progress through `examples/002_agent_communication.py` to `examples/009_emergent_collective_intelliger`
   - Try `examples/venturelens.py` for a complete real-world application
2. **Build Your First System**:
   - Start with 2-3 specialized agents
   - Focus on clear agent identities and communication
   - Let behaviors emerge from agent interactions
3. **Add Advanced Features**:
   - Enable persistent memory with Qdrant
   - Implement secure communication for sensitive data
   - Deploy with Docker for production environments
4. **Scale Up**:
   - Introduce new specialists as needed
   - Use registry patterns for agent discovery
   - Monitor system health and performance

Learn More

- **Complete Manual**: `praval.md` - Comprehensive guide with examples
- **Examples**: `examples/` directory - Progressive learning journey
- **Documentation**: In-depth technical references
- **Community**: Join the coral reef of Praval developers

"In simplicity lies the ultimate sophistication. In collaboration lies the future of intelligence."

Welcome to the age of collaborative AI. Welcome to Praval.