MoleculeInsight – AI Agent Research Platform

Complete Technical Documentation

Table of Contents

Overview

MoleculeInsight is an advanced AI-powered platform designed for comprehensive molecular research and pharmaceutical innovation scouting. The platform addresses the critical challenge faced by pharmaceutical companies: **molecular research that traditionally takes 2-3 months can now be completed in minutes**.

Problem Statement

Generic pharmaceutical companies face intense competition and low profit margins. To create innovative products, scientists must:

- Read hundreds of research papers
- Check clinical trial databases

- Analyze patent landscapes
- Study market dynamics
- Search multiple web sources
- Review internal documents

This manual process is **slow, repetitive, and time-consuming**.

Solution

MoleculeInsight orchestrates a team of specialized AI agents that work in parallel to:

- Search clinical trials (ClinicalTrials.gov)
- Analyze patent landscapes (PatentsView)
- Track global trade flows (UN Comtrade)
- Gather market intelligence (IQVIA insights)
- Monitor web news and publications
- Query internal knowledge bases via RAG
- Synthesize findings into actionable innovation strategies

Result: Complete molecular evaluation in minutes instead of months.

System Architecture

High-Level Architecture

```
┌─────────────────────────────────────────────────────────────┐
│                    Frontend (Next.js)                       │
│  ┌──────────────┐  ┌──────────────┐  ┌──────────────┐      │
│  │  Dashboard   │  │   Analysis   │  │   Reports    │      │
│  │              │  │  Interface   │  │              │      │
│  └──────────────┘  └──────────────┘  └──────────────┘      │
└─────────────────────────────────────────────────────────────┘
                          │
                          │ HTTP/WebSocket
                          ▼
┌─────────────────────────────────────────────────────────────┐
│                   Backend API (FastAPI)                     │
│                              │
│ ┌─────────────────────────────────────────────────────────┐ │
│ │        Master Orchestration Agent            ││
│ │ (Gemini AI - Task Decomposition & Synthesis)     ││
│ └─────────────────────────────────────────────────────────┘ │
│              │                 │
│    ┌───────────────────────┼──────────────────────┐       │
│    ▼         ▼         ▼       │
│ ┌──────────────┐ ┌────────────┐  ┌────────────┐      │
│ │ Worker  │  │ Worker  │  │ Worker  │   │
│ │ Agent 1 │  │ Agent 2 │  │ Agent N │   │
│ └──────────────┘ └────────────┘  └────────────┘      │
│    │       │       │      │
└─────────────┼───────────────────┼───────────────────┼─────────────────┘
     │       │       │
     ▼       ▼       ▼
┌────────────────┐ ┌────────────────┐ ┌────────────────┐
│ ClinicalTrials│ │ PatentsView │ │ UN Comtrade │
│    .gov    │ │    API   │ │    API   │
└────────────────┘ └────────────────┘ └────────────────┘

     ▼       ▼       ▼
┌────────────────┐ ┌────────────────┐ ┌────────────────┐
│  NewsAPI   │ │ Web Search │ │ ChromaDB RAG │
│        │ │    Proxy   │ │  (Internal) │
└────────────────┘ └────────────────┘ └────────────────┘
```

Data Flow

1. **User Input Capture**
   - User enters research query in web interface
   - Query sent to FastAPI backend via `/api/analyze` endpoint
2. **Master Agent Processing**
   - Master Orchestration Agent (Gemini AI) receives query
   - Breaks down query into specialized subtasks:
     - Clinical trial analysis
     - Patent landscape review
     - Market intelligence gathering
     - Trade flow analysis

        - Web intelligence search
        - Internal knowledge lookup

3. **Parallel Worker Execution**
    - Worker agents execute concurrently using `concurrent.futures`
    - Each agent connects to its designated data source
    - Returns structured JSON results
4. **Data Aggregation**
    - Master Agent collects all worker outputs
    - Performs cross-source analysis and synthesis
    - Generates innovation strategy recommendations
5. **Dashboard Rendering**
    - Frontend receives structured data
    - Displays interactive cards for each data source
    - Presents AI-generated innovation concept
6. **Report Generation**
    - Innovation Strategy Agent compiles final report
    - Exports as PDF/Markdown with all findings

Key Features

🤖 Multi-Agent Orchestration

Coordinated System of Specialized Agents

- Master Agent for task decomposition and synthesis
- 7 specialized Worker Agents operating in parallel
- Real-time status updates via WebSocket connections
- Fault-tolerant design with fallback mechanisms

🧠 RAG-Powered Knowledge System

Retrieval-Augmented Generation (RAG)

- Vector database: **ChromaDB**
- Supports: JSON, PDF, text documents
- Embedding model: Google Gemini Embeddings
- Enables context-aware responses from internal data
- Query expansion and semantic search

⚡ Real-time Analysis

Live Monitoring & Progress Tracking

- WebSocket-based agent status updates
- Progress bar showing completion percentage
- Individual agent activity indicators
- Estimated time remaining calculations

📊 Comprehensive Data Sources

Data Source	**Agent**	**Information Captured**
ClinicalTrials.gov	Clinical Trials Agent	Trial phases, sponsors, indications, enrollment status
PatentsView	Patent Agent	Active patents, expiry dates, FTO risks, filing trends
UN Comtrade	EXIM Trade Agent	Import/export volumes, API dependencies, trade flows
IQVIA (Mock)	Market Intelligence Agent	Market size, CAGR, competition, unmet needs
NewsAPI	Web Intelligence Agent	Recent publications, guidelines, industry news
Internal Docs	RAG Knowledge Agent	Proprietary research, past projects, expertise

📈 Interactive Dashboard

Rich Visualizations

- Market trend charts (Recharts)
- Patent timeline visualizations
- Clinical trial phase distributions
- Trade flow heat maps

- AI-generated innovation concept card

📝 Automated Reporting

One-Click Export

- Markdown format for easy sharing
- PDF generation with charts and tables
- Includes all agent findings
- AI-synthesized executive summary
- Archival system for historical queries

Technology Stack

Frontend

Technology	Version	Purpose
Next.js	16.x	React framework with App Router
React	19.x	UI component library
TypeScript	5.x	Type-safe development
Tailwind CSS	3.x	Utility-first styling
Shadcn UI	Latest	Premium component library
Radix UI	Latest	Accessible primitives
Recharts	2.x	Data visualization
Framer Motion	Latest	Animations
Lucide React	Latest	Icon system

Backend

Technology	Version	Purpose
Python	3.9+	Core language
FastAPI	0.100+	REST API framework
Google Gemini AI	Latest	LLM for orchestration
ChromaDB	0.4+	Vector database for RAG
asyncio	Built-in	Asynchronous operations
concurrent.futures	Built-in	Parallel agent execution
Pydantic	2.x	Data validation
httpx	0.24+	HTTP client for API calls

External APIs

- **ClinicalTrials.gov API** - Trial data
- **PatentsView API** - Patent information
- **UN Comtrade API** - Trade statistics
- **NewsAPI** - Web intelligence
- **Google Gemini API** - AI orchestration

Installation Guide

Prerequisites

System Requirements:

- Node.js v18 or higher
- Python 3.9 or higher
- 8GB RAM minimum (16GB recommended)
- 2GB free disk space

Required API Keys:

- Google Gemini API Key (required)
- UN Comtrade API Key (optional)
- NewsAPI Key (optional)

Step 1: Clone Repository

```bash
git clone https://github.com/BikramMondal5/MoleculeInsight.git
cd MoleculeInsight
```

Step 2: Backend Setup

```bash
# Navigate to agents directory
cd agents

# Create virtual environment
python -m venv venv

# Activate virtual environment
# Windows (PowerShell):
.\venv\Scripts\Activate.ps1

# macOS/Linux:
source venv/bin/activate

# Install Python dependencies
pip install -r requirements.txt
```

requirements.txt contents:

```txt
```

```
fastapi==0.100.0
uvicorn[standard]==0.23.0
google-generativeai==0.3.0
chromadb==0.4.0
pydantic==2.0.0
httpx==0.24.0
python-dotenv==1.0.0
PyPDF2==3.0.0
python-multipart==0.0.6
```

Step 3: Frontend Setup

```bash
# Return to project root
cd ..

# Install Node.js dependencies (using pnpm)
pnpm install
```

If you don't have pnpm:

```bash
npm install -g pnpm
pnpm install
```

Step 4: Environment Configuration

Create `.env` file in `agents/RAG/.env`:

```env
```

```
# Google Gemini API Key (Required)
GOOGLE_API_KEY=your_gemini_api_key_here

# Optional API Keys
NEWS_API_KEY=your_newsapi_key
COMTRADE_KEY=your_comtrade_key

# RAG Configuration
CHROMA_PERSIST_DIR=./db
COLLECTION_NAME=molecule_knowledge

# Agent Configuration
MAX_WORKERS=6
TIMEOUT_SECONDS=30
```

Step 5: RAG Knowledge Base Setup

```bash
# Navigate to RAG directory
cd agents/RAG

# Place your documents in KnowledgeBase folder
# Supported formats: .json, .pdf, .txt

# Run ingestion script
python ingest_all.py
```

Expected Output:

```
Processing documents...
✓ Loaded 15 JSON files
✓ Loaded 8 PDF files
✓ Created 2,450 embeddings
✓ Stored in ChromaDB collection: molecule_knowledge
Ingestion complete!
```

Configuration

Backend Configuration

agents/config.py:

```python
from pydantic import BaseSettings

class Settings(BaseSettings):
    # API Configuration
    API_HOST: str = "0.0.0.0"
    API_PORT: int = 8000

    # Google Gemini
    GOOGLE_API_KEY: str
    GEMINI_MODEL: str = "gemini-1.5-pro"

    # RAG Configuration
    CHROMA_PERSIST_DIR: str = "./RAG/db"
    COLLECTION_NAME: str = "molecule_knowledge"

    # Agent Configuration
    MAX_WORKERS: int = 6
    AGENT_TIMEOUT: int = 30

    # External APIs
    CLINICAL_TRIALS_API: str = "https://clinicaltrials.gov/api/v2"
    PATENTS_VIEW_API: str = "https://api.patentsview.org"
    COMTRADE_API: str = "https://comtradeapi.un.org"

    class Config:
        env_file = "RAG/.env"

settings = Settings()
```

Frontend Configuration

next.config.js:

```javascript
```

```
/** @type {import('next').NextConfig} */
const nextConfig = {
  reactStrictMode: true,
  env: {
    NEXT_PUBLIC_API_URL: process.env.NEXT_PUBLIC_API_URL || 'http://localhost:8000',
  },
  images: {
    domains: ['localhost'],
  },
}

module.exports = nextConfig
```

.env.local:

```
env

NEXT_PUBLIC_API_URL=http://localhost:8000
```

Agent System

Master Orchestration Agent

Location: `agents/orchestrator.py`

Responsibilities:

1. Parse user query and identify research intent
2. Break query into specialized subtasks
3. Dispatch tasks to Worker Agents
4. Collect and aggregate results
5. Synthesize findings using Gemini AI
6. Generate innovation strategy

Key Functions:

```
python
```

```python
class MasterAgent:
    def __init__(self):
        self.gemini_model = genai.GenerativeModel('gemini-1.5-pro')
        self.workers = self._initialize_workers()

    async def analyze_molecule(self, query: str, filters: dict):
        """Main orchestration method"""
        # 1. Task decomposition
        tasks = self._decompose_query(query)

        # 2. Parallel execution
        results = await self._execute_workers(tasks)

        # 3. Synthesis
        innovation = await self._synthesize_findings(results)

        return {
            "results": results,
            "innovation_concept": innovation
        }
```

Worker Agents

1. Clinical Trials Agent

File: `agents/Agent-workers/clinical_trials_agent.py`

Data Source: ClinicalTrials.gov API

Captured Data:

- Trial phases (I, II, III, IV)
- Enrollment status (recruiting, completed, terminated)
- Sponsors and collaborators
- Indications and outcomes
- Study design details

Sample Code:

```
python
```

```
class ClinicalTrialsAgent:
    async def search_trials(self, molecule_name: str):
        url = f"{CLINICAL_TRIALS_API}/studies"
        params = {
            "query.term": molecule_name,
            "fields": "NCTId,BriefTitle,OverallStatus,Phase"
        }
        response = await self.http_client.get(url, params=params)
        return self._parse_trials(response.json())
```

2. Patent Landscape Agent

File: `agents/Agent-workers/patent_agent.py`

Data Source: PatentsView API / USPTO

Captured Data:

- Active patents and expiry dates
- Freedom-to-Operate (FTO) assessment
- Patent families and citations
- Inventor and assignee information
- Technology classification

3. EXIM Trade Agent

File: `agents/Agent-workers/exim_agent.py`

Data Source: UN Comtrade API

Captured Data:

- Import/export volumes by country
- API (Active Pharmaceutical Ingredient) dependencies
- Trade flow trends
- Regulatory compliance risks

4. Market Intelligence Agent

File: `agents/Agent-workers/market_agent.py`

Data Source: Mock IQVIA data (can be integrated with real API)

Captured Data:

- Market size and CAGR
- Competitive landscape
- Unmet medical needs
- Therapeutic area trends

5. Web Intelligence Agent

File: `agents/Agent-workers/web_agent.py`

Data Sources: NewsAPI, Web Search Proxy

Captured Data:

- Recent publications
- Regulatory guidelines (FDA, EMA)
- Industry news and announcements
- Conference proceedings

6. Internal Knowledge Agent

File: `agents/Agent-workers/internal_agent.py`

Data Source: ChromaDB RAG System

Captured Data:

- Past project summaries
- Internal expertise availability
- Manufacturing capabilities
- Regulatory track record

7. Innovation Strategy Agent

File: `agents/Agent-workers/innovation_agent.py`

Purpose: Final synthesis agent

Responsibilities:

- Cross-reference all data sources
- Identify opportunity gaps
- Assess feasibility
- Generate innovation recommendations
- Create executive summary

RAG Knowledge Engine

Architecture

ChromaDB Vector Database

- Persistent storage: `agents/RAG/db/`
- Collection: `molecule_knowledge`
- Embedding model: Google Gemini Embeddings
- Distance metric: Cosine similarity

Document Ingestion Pipeline

Location: `agents/RAG/ingest_all.py`

Process:

1. **Document Loading**

```python
def load_documents(knowledge_base_path):
    documents = []
    for file in os.listdir(knowledge_base_path):
        if file.endswith('.json'):
            documents.extend(load_json(file))
        elif file.endswith('.pdf'):
            documents.extend(load_pdf(file))
    return documents
```

2. **Text Chunking**

```python
def chunk_documents(documents, chunk_size=500):
    chunks = []
    for doc in documents:
        text = doc['content']
        for i in range(0, len(text), chunk_size):
            chunks.append({
                'text': text[i:i+chunk_size],
                'metadata': doc['metadata']
            })
    return chunks
```

3. **Embedding Generation**

```python
def generate_embeddings(chunks):
    embeddings = []
    for chunk in chunks:
        embedding = genai.embed_content(
            model="models/embedding-001",
            content=chunk['text']
        )
        embeddings.append(embedding)
    return embeddings
```

4. **Vector Storage**

```python
def store_in_chromadb(chunks, embeddings):
    collection = chroma_client.create_collection(
        name="molecule_knowledge"
    )
    collection.add(
        documents=[c['text'] for c in chunks],
        embeddings=embeddings,
        metadatas=[c['metadata'] for c in chunks],
        ids=[str(i) for i in range(len(chunks))]
    )
```

Query Process

Location: `agents/RAG/query_rag.py`

```python
async def query_knowledge_base(query: str, top_k: int = 5):
    # 1. Generate query embedding
    query_embedding = genai.embed_content(
        model="models/embedding-001",
        content=query
    )

    # 2. Search ChromaDB
    results = collection.query(
        query_embeddings=[query_embedding],
        n_results=top_k
    )

    # 3. Rerank results
    reranked = rerank_by_relevance(results)

    # 4. Format context
    context = format_for_llm(reranked)

    return context
```

Supported Document Formats

JSON Files:

```json
{
  "project_id": "PRJ-2024-001",
  "molecule": "Compound X",
  "indication": "Respiratory",
  "status": "Completed",
  "findings": "..."
}
```

PDF Files:

- Research reports

- Patent documents
- Internal memos
- Study protocols

API Reference

Base URL

```
http://localhost:8000
```

Endpoints

1. Analyze Molecule

POST `/api/analyze`

Description: Initiates multi-agent analysis for a molecule

Request Body:

```json
{
  "query": "Find repurposing opportunities for Aspirin in cardiovascular diseases",
  "filters": {
    "geography": ["USA", "EU"],
    "therapeutic_areas": ["Cardiology"],
    "include_patents": true,
    "include_trials": true
  }
}
```

Response:

```json
```

```
{
  "status": "success",
  "analysis_id": "AN-20240512-001",
  "progress": 100,
  "results": {
    "market": {...},
    "patents": {...},
    "trials": {...},
    "exim": {...},
    "web": {...},
    "internal": {...}
  },
  "innovation_concept": "Based on comprehensive analysis..."
}
```

2. Get Analysis Status

GET `/api/analysis/{analysis_id}/status`

Response:

```json
{
  "analysis_id": "AN-20240512-001",
  "status": "in_progress",
  "progress": 65,
  "active_agents": ["patent_agent", "web_agent"],
  "completed_agents": ["clinical_trials", "market", "exim"]
}
```

3. Download Report

GET `/api/analysis/{analysis_id}/report`

Query Parameters:

- `format`: `pdf` or `markdown`

Response: Binary file download

4. Query RAG System

POST `/api/rag/query`

Request Body:

```json
{
  "query": "What was the outcome of our previous respiratory project?",
  "top_k": 5
}
```

Response:

```json
{
  "results": [
    {
      "text": "Project PRJ-2023-045 successfully...",
      "metadata": {...},
      "score": 0.89
    }
  ]
}
```

5. WebSocket Connection

WS `/ws/analysis/{analysis_id}`

Messages:

```json
{
  "type": "agent_status",
  "agent": "clinical_trials_agent",
  "status": "working",
  "progress": 35
}
```

User Guide

Getting Started

1. **Access Dashboard**
    - Navigate to `http://localhost:3000/analysis`
2. **Enter Query**
    - Type your molecule research question
    - Examples:
        - "Analyze market potential for Metformin in diabetes"
        - "Find repurposing opportunities for Aspirin"
        - "What are the patent barriers for Ibuprofen formulations?"
3. **Configure Filters (Optional)**
    - Select target geographies
    - Choose therapeutic areas
    - Enable/disable specific data sources
4. **Run Analysis**
    - Click "Run Agentic Analysis"
    - Watch real-time agent activity

Understanding Results

Market Insights Card

- **Market Size:** Current market value
- **CAGR:** Growth rate projection
- **Unmet Needs:** Identified gaps in current offerings
- **Competition:** Key players

Patent Landscape Card

- **Active Patents:** Number of blocking patents
- **Expiring Soon:** Opportunities from patent expiry
- **FTO Risk:** Freedom-to-Operate assessment

- **Opportunities:** Clear pathways for innovation

Clinical Trials Card

- **Phase Distribution:** Breakdown by trial phase
- **Ongoing Trials:** Current research activity
- **Gap Areas:** Under-researched indications

Innovation Concept

- AI-synthesized strategic recommendation
- Highlights key differentiators
- Assesses feasibility
- Estimates timeline

Exporting Results

1. **Download Report:**
    - Click "Download Full Report"
    - Choose format (PDF/Markdown)
    - Report saved to Downloads folder
2. **Share Analysis:**
    - Copy shareable link
    - Set access permissions
    - Expires after 30 days

Deployment

Production Deployment (Docker)

docker-compose.yml:

```
yaml
```

```yaml
version: '3.8'

services:
 backend:
  build: ./agents
  ports:
   - "8000:8000"
  environment:
   - GOOGLE_API_KEY=${GOOGLE_API_KEY}
  volumes:
   - ./agents/RAG/db:/app/RAG/db

 frontend:
  build: .
  ports:
   - "3000:3000"
  environment:
   - NEXT_PUBLIC_API_URL=http://backend:8000
  depends_on:
   - backend
```

Deploy:

```bash
docker-compose up -d
```

Cloud Deployment (AWS)

Architecture:

- Frontend: AWS Amplify / Vercel
- Backend: AWS ECS / EC2
- Database: AWS RDS (if needed)
- Storage: S3 (for ChromaDB persistence)

Steps:

1. Build Docker images
2. Push to AWS ECR
3. Deploy ECS service

4. Configure load balancer
5. Set up CloudWatch monitoring

Troubleshooting

Common Issues

1. "ChromaDB collection not found"

```bash
# Solution: Re-run ingestion
cd agents/RAG
python ingest_all.py
```

2. "Gemini API quota exceeded"

```
Error: 429 Too Many Requests
```

Solution:

- Check API quota limits
- Implement rate limiting
- Use exponential backoff

3. "Agent timeout errors"

```
TimeoutError: Clinical trials agent exceeded 30s
```

Solution:

- Increase `AGENT_TIMEOUT` in config
- Check network connectivity
- Verify API endpoint availability

4. "Frontend can't connect to backend"

```
Error: ECONNREFUSED 127.0.0.1:8000
```

Solution:

- Ensure backend is running: `python main.py`
- Check `NEXT_PUBLIC_API_URL` in `.env.local`
- Verify firewall settings

Contributing

Development Workflow

1. **Fork Repository**
2. **Create Feature Branch**

```bash
git checkout -b feature/new-agent
```

3. **Make Changes**
4. **Test Locally**

```bash
pytest agents/tests/
```

5. **Submit Pull Request**

Adding New Agents

Template: `agents/Agent-workers/template_agent.py`

```python
```

```python
class NewAgent:
    def __init__(self):
        self.name = "new_agent"

    async def execute(self, query: str):
        """Main execution method"""
        try:
            # 1. API call or data fetch
            data = await self._fetch_data(query)

            # 2. Process results
            processed = self._process(data)

            # 3. Return structured output
            return {
                "status": "success",
                "data": processed
            }
        except Exception as e:
            return {
                "status": "error",
                "message": str(e)
            }
```

Code Style

- **Python:** Follow PEP 8
- **JavaScript:** ESLint + Prettier
- **Commits:** Conventional Commits format

License

This project is licensed under the **MIT License**.

Support

- **GitHub Issues:** https://github.com/BikramMondal5/MoleculeInsight/issues
- **Email:** support@moleculeinsight.com
- **Documentation:** https://docs.moleculeinsight.com

Built with ❤️ for pharmaceutical innovation

Last Updated: December 2024