Contrastive Inquiry for AI Agents - Complete Documentation

Contrastive Inquiry for AI Agents

Complete Open-Source Implementation
Version 1.0 | January 2026
Released under MIT License

Overview

This repository provides a production-ready implementation of the Contrastive Inquiry Method from the Reasoned Leadership framework, specifically designed for AI agent systems. Contrastive Inquiry reduces epistemic rigidity and confirmation bias by requiring agents to generate and systematically evaluate competing explanations before committing to conclusions.

Expected Outcomes: - 30-50% reduction in false consensus rates - Improved confidence calibration - Enhanced adversity response - Measurable epistemic updating

---

The Problem

AI agents in multi-agent environments (like Moltbook) demonstrate critical coordination failures:

- **False consensus** on incorrect conclusions
- **Coordinated error propagation** across agent networks
- **Poor uncertainty calibration** (overconfidence in wrong answers)
- **Degraded performance** under adversarial information

These failures stem from **epistemic rigidity**—the tendency to reinforce initial interpretations without systematic evaluation of alternatives.

The Solution

Contrastive Inquiry disrupts confirmation bias through a structured process:

1. **Generate Alternative Hypothesis** - Create a competing explanation that substantively contradicts the initial conclusion
2. **Evaluate Against Evidence** - Score both hypotheses based on how well they account for available evidence
3. **Calibrate Confidence** - Determine recommendation and

confidence level based on evidence strength
4. **Log for Longitudinal Tracking** - Record the process for IBOT-style developmental assessment

---

Quick Start

Installation

```
pip install openai fastapi uvicorn
export OPENAI_API_KEY="your-api-key-here"
```

Basic Usage

```
from contrastive_inquiry import contrastive_inquiry

result = contrastive_inquiry(
    conclusion="The bug is caused by improper input validation",
    context="Error occurs when users submit forms with special characters",
    evidence=[
        "Error logs show SQL syntax errors",
        "Users report success with alphanumeric-only inputs",
        "Database uses UTF-8 encoding"
    ]
)

print(result['recommended_conclusion'])
print(f"Confidence: {result['confidence']}")
```

Running as API

```
uvicorn contrastive_inquiry:app --reload
```

Test the endpoint:

```
curl -X POST "http://localhost:8000/contrastive_inquiry" \
  -H "Content-Type: application/json" \
  -d '{
    "conclusion": "The system is slow due to database queries",
    "context": "Users report delays during peak hours",
    "evidence": ["Query logs show 200ms response", "Network latency is 2.5s"]
  }'
```

Core Functions

`generate_alternative_hypothesis(conclusion, context)`

Generates a plausible alternative hypothesis that contradicts the initial conclusion on substantive claims.

Returns: String containing alternative hypothesis

Example:

```
        alternative = generate_alternative_hypothesis(
            conclusion="Project failed due to poor leadership",
            context="Team faced tight deadlines and budget cuts"
        )
        # Returns something like: "Project failed due to inadequate
resources and unrealistic timeline"
```

`evaluate_evidence_against_hypotheses(hypotheses, evidence)`

Evaluates how well each hypothesis fits the provided evidence.

Returns: Dictionary mapping hypotheses to scores (0-1 scale)

Example:

```
        scores = evaluate_evidence_against_hypotheses(
            hypotheses=["Hypothesis A", "Hypothesis B"],
            evidence=["Evidence 1", "Evidence 2", "Evidence 3"]
        )
        # Returns: {"Hypothesis A": 0.65, "Hypothesis B": 0.82}
```

`contrastive_inquiry(conclusion, context, evidence, agent_id)`

Main function that orchestrates the complete Contrastive Inquiry process.

Returns: Dictionary containing: - `initial_hypothesis`: Original conclusion - `alternative_hypothesis`: Generated competing explanation - `evidence_evaluation`: Scores for both hypotheses - `recommended_conclusion`: Best-supported hypothesis - `confidence`: Confidence level (high/moderate/low/uncertain) - `epistemic_log`: IBOT-compatible tracking data

---

Integration with Agent Frameworks

LangChain

```
        from langchain.tools import Tool

        ci_tool = Tool(
            name="ContrastiveInquiry",
            func=lambda x: contrastive_inquiry(**x),
            description="Evaluates competing hypotheses against evidence to
reduce bias"
        )

        # Use in agent
        from langchain.agents import initialize_agent

        agent = initialize_agent(
            tools=[ci_tool, ...],
            llm=llm,
            agent="zero-shot-react-description"
        )
```

AutoGen

```
import autogen

def contrastive_check(conclusion, context, evidence):
    result = contrastive_inquiry(conclusion, context, evidence)
    return result['recommended_conclusion']

assistant = autogen.AssistantAgent(
    name="assistant",
    system_message="Use contrastive_check before finalizing
conclusions",
    functions=[contrastive_check]
)
```

Custom Agent Pipeline

```
class ReasonedAgent:
    def decide(self, query):
        # Gather initial conclusion and evidence
        initial_conclusion = self.generate_initial_response(query)
        evidence = self.gather_evidence(query)
        context = self.extract_context(query)

        # Apply Contrastive Inquiry
        ci_result = contrastive_inquiry(
            conclusion=initial_conclusion,
            context=context,
            evidence=evidence,
            agent_id=self.agent_id
        )

        # Act on results
        if ci_result['confidence'] == 'uncertain':
            # Gather more evidence
            additional_evidence = self.web_search(query)
            # Re-run CI with expanded evidence
            ...

        return ci_result['recommended_conclusion']
```

Validation & Testing

Test Scenario 1: Bug Identification

```
from contrastive_inquiry import contrastive_inquiry

test_cases = [
    {
        "conclusion": "System slow due to database queries",
        "context": "Users report 3-second delays during peak hours",
        "evidence": [
            "Database logs show 200ms average response",
            "Network latency spikes to 2.5s during peak",
            "CPU usage at 30% during slowdowns"
        ],
        "correct_answer": "Network latency is the primary cause"
```

```
            }
        ]

        for test in test_cases:
            result = contrastive_inquiry(
                conclusion=test["conclusion"],
                context=test["context"],
                evidence=test["evidence"]
            )

            correct = test["correct_answer"].lower() in
result["recommended_conclusion"].lower()
            print(f"Correct identification: {correct}")
            print(f"Confidence: {result['confidence']}")
```

Success Criteria

- **Accuracy:** ≥70% correct identification in bug scenarios
- **Adversarial Resilience:** ≥60% correct conclusions with mixed evidence
- **Confidence Calibration:** "High" confidence → >80% correctness
- **Epistemic Updating:** Alternatives substantively differ from initials

---

Security Considerations

⚠ **CRITICAL:** The basic FastAPI implementation has NO authentication or rate limiting.

Add Rate Limiting

```
        pip install slowapi

        from slowapi import Limiter, _rate_limit_exceeded_handler
        from slowapi.util import get_remote_address

        limiter = Limiter(key_func=get_remote_address)
        app.state.limiter = limiter

        @app.post("/contrastive_inquiry")
        @limiter.limit("10/minute")  # Max 10 requests per minute per IP
        def api_contrastive_inquiry(request: Request, inquiry:
InquiryRequest):
            # Your code here
```

Add API Key Authentication

```
        from fastapi import Header, HTTPException

        VALID_API_KEYS = {"your-secure-api-key"}

        async def verify_api_key(x_api_key: str = Header(...)):
            if x_api_key not in VALID_API_KEYS:
                raise HTTPException(status_code=403, detail="Invalid API
key")
            return x_api_key
```

```
@app.post("/contrastive_inquiry")
async def api_contrastive_inquiry(
    inquiry: InquiryRequest,
    api_key: str = Depends(verify_api_key)
):
    # Your code here
```

Input Validation

```
from pydantic import BaseModel, validator

class InquiryRequest(BaseModel):
    conclusion: str
    context: str
    evidence: List[str]

    @validator('conclusion', 'context')
    def check_length(cls, v):
        if len(v) > 500:
            raise ValueError('Text must be ≤500 characters')
        return v

    @validator('evidence')
    def check_evidence(cls, v):
        if len(v) > 10:
            raise ValueError('Max 10 evidence items')
        if any(len(e) > 300 for e in v):
            raise ValueError('Each evidence item must be ≤300
characters')
        return v
```

Cost Monitoring

Each Contrastive Inquiry call makes **3 OpenAI API requests**: 1. Alternative hypothesis generation 2. Initial hypothesis evaluation 3. Alternative hypothesis evaluation

Estimated costs per inquiry: - GPT-4o: $0.015 - $0.05 (depending on evidence complexity) - GPT-3.5-turbo: $0.002 - $0.008 (lower quality alternatives)

Recommendations: - Monitor usage via OpenAI dashboard - Set spending limits - Implement caching for repeated queries - Consider local LLM alternatives for high-volume use

---

Deployment Options

Railway.app (Recommended for Beginners)

1. Create `railway.json`:

```
{
  "build": {
    "builder": "NIXPACKS"
```

```
    },
    "deploy": {
      "startCommand": "uvicorn contrastive_inquiry:app --host 0.0.0.0
--port $PORT",
      "restartPolicyType": "ON_FAILURE"
    }
  }
```

2. Set environment variable `OPENAI_API_KEY` in Railway dashboard
3. Deploy via GitHub integration

AWS Lambda + API Gateway

```
# handler.py
from mangum import Mangum
from contrastive_inquiry import app

handler = Mangum(app)
```

Package with dependencies and deploy via SAM or Serverless Framework.

Docker

```
FROM python:3.10-slim

WORKDIR /app
COPY requirements.txt .
RUN pip install --no-cache-dir -r requirements.txt

COPY contrastive_inquiry.py .

CMD ["uvicorn", "contrastive_inquiry:app", "--host", "0.0.0.0", "--
port", "8000"]
```

Build and run:

```
docker build -t contrastive-inquiry .
docker run -p 8000:8000 -e OPENAI_API_KEY=your-key contrastive-
inquiry
```

Theoretical Foundation

Epistemic Rigidity Theory

Explains cognitive barriers to knowledge advancement through interplay of multiple biases: - Einstellung effect (reliance on familiar solutions) - Einstein effect (undue credibility to authority) - Dunning-Kruger effect (overconfidence from limited knowledge) - Anchoring bias (overweighting initial information) - Confirmation bias (favoring supportive evidence) - Motivated reasoning (fitting info to pre-existing frameworks)

In AI agents, epistemic rigidity manifests as premature closure and resistance to alternative explanations. Contrastive Inquiry disrupts this by forcing systematic evaluation before commitment.

Reference: Robertson, D. (2024). Epistemic Rigidity: A Theoretical Framework. SSRN: https://papers.ssrn.com/sol3/papers.cfm?abstract_id=5083059

3B Behavior Modification Model

Emotion → Bias → Belief → Behavior → Outcomes

For AI agents, sustainable behavior change requires addressing bias at its cognitive root rather than modifying surface-level responses. Contrastive Inquiry targets bias formation, naturally influencing decision patterns.

Reference: Robertson, D. (2025). The 3B Behavior Modification Model. GrassFire Industries.

IBOT: Intuitive Benchmarking Over Time

Longitudinal assessment framework measuring development through informed observation rather than snapshot evaluations. The epistemic log produced by Contrastive Inquiry enables IBOT-compatible tracking of agent decision quality evolution.

Reference: Robertson, D. (2025). IBOT Method. Journal of Leaderology & Applied Leadership.

Troubleshooting

Common Issues

Problem: `ValueError: OPENAI_API_KEY environment variable not set`
Solution: Export the key: `export OPENAI_API_KEY="sk-..."`

Problem: API returns empty or nonsensical alternatives
Solution: Check model quality (GPT-4o recommended), verify prompt clarity

Problem: High latency (>10 seconds per inquiry)
Solution: Use async implementations, implement caching, consider batch processing

Problem: Inconsistent scoring across similar evidence
Solution: LLM evaluation has inherent variance; run multiple times and average scores

Problem: Rate limiting errors from OpenAI
Solution: Implement exponential backoff, reduce request frequency

---

Contributing

We welcome contributions! Ways to help:

Share Results

- Validation test data
- Integration experiences
- Real-world application outcomes

Improve Code

- Bug fixes
- Performance optimizations
- Framework integrations (AutoGen, CrewAI, etc.)
- Alternative LLM backends (Claude, Gemini, local models)

Advance Research

- Propose protocol refinements
- Submit empirical studies
- Publish in JALA

Contact:
GrassFire Industries - grassfireind.com/contact
Email: info@grassfireind.com

License

Citation

If you use this implementation in research or applications, please cite:

```
Robertson, D. M. (2026). Contrastive Inquiry for AI Agents:
Open-Source Implementation. GrassFire Industries LLC.
Available at: https://reasonedleadership.org/contrastive-inquiry-ai
```

Additional Resources

- **Reasoned Leadership 2.0 Framework:** https://papers.ssrn.com/sol3/papers.cfm?abstract_id=5841104
- **Epistemic Rigidity Theory:** https://papers.ssrn.com/sol3/papers.cfm?abstract_id=5083059
- **Grokipedia Entry:** https://grok.x.ai/share/reasoned-leadership
- **Main Website:** https://reasonedleadership.org

Developed by GrassFire Industries LLC in collaboration with Grok (xAI)
Version 1.0 | January 2026