

# AVA PROTOCOL

## Complete Usage Examples

*All Operating Modes | All Vault Types | All Deployment Patterns*

1. Embedded Mode (Local Presidio)
2. Gateway Mode (Remote Client)
3. Mock Engine (Testing)
4. AWS Macie Adapter
5. Azure PII Adapter
6. Google DLP Adapter
7. Memory/SQLite/Redis Vaults
8. Policy Configurations
9. Async API
10. Production Workflows

**Gerald Enrique Nelson Mc Kenzie**

PyPI: [pypi.org/project/ava-protocol/](https://pypi.org/project/ava-protocol/)

# Installation Matrix

---

Choose installation based on which modes you need:

## Installation Commands

```
# MODE 1: Gateway Client Only (Lightweight)
pip install ava-protocol
# Size: ~50KB | Use: Connect to remote AVA Gateway

# MODE 2: Embedded with Presidio (Full Local)
pip install ava-protocol[local]
# Size: ~500MB | Use: Self-contained PII detection

# MODE 3: AWS Integration
pip install ava-protocol[aws]
# Adds: boto3 | Use: AWS Macie detection

# MODE 4: Azure Integration
pip install ava-protocol[azure]
# Adds: azure-ai-textanalytics

# MODE 5: Google Cloud Integration
pip install ava-protocol[gcp]
# Adds: google-cloud-dlp

# MODE 6: Enterprise (Everything)
pip install ava-protocol[all]
# Includes: local + aws + azure + gcp + redis
```

**NOTE:** Gateway mode requires NO extras. Embedded mode requires [local] for Presidio ML models.

## MODE: 1. EMBEDDED MODE (Full Local)

Self-contained deployment. All PII detection happens locally. Best for air-gapped environments.

### Prerequisites

#### Install

```
pip install ava-protocol[local]
```

### Basic Example

#### embedded\_basic.py

```
import ava

# Create client with local Presidio engine
client = ava.Client(
    engine="presidio",
    policy="healthcare_strict",
    vault_type="memory"
)

# Process sensitive text
with client.session(reversibility=True, ttl=3600) as session:

    medical_record = '''
    Patient: Maria Gonzalez
    DOB: 1985-03-15
    SSN: 123-45-6789
    Email: maria.g@healthmail.com
    Diagnosis: Hypertension
    '''

    # Sanitize before AI processing
    sanitized = session.sanitize(medical_record)
    print(sanitized)
    # Patient: AVA_PERS_xK9mP2nQ
    # DOB: AVA_DATE_aB3cD4eF
    # SSN: AVA_SSN_fG5hI6jK

    # Send to OpenAI (AI never sees real data)
    import openai
    response = openai.ChatCompletion.create(
        model="gpt-4",
        messages=[{"role": "user", "content": sanitized}]
    )

    # Restore original values
    final = session.restore(response['choices'][0]['message']['content'])
    print(final) # Original names restored!
```

## MODE: 2. GATEWAY MODE (Remote Client)

Thin client connecting to remote AVA Gateway. No local ML dependencies.

### Prerequisites

#### Install

```
pip install ava-protocol # No extras needed!
```

### Basic Gateway Client

#### gateway\_client.py

```
import ava

# Connect to centralized AVA Gateway
client = ava.Client(
    gateway_url="https://ava-gateway.company.com",
    api_key="ava_sk_live_abc123xyz789",
    policy="general_moderate"
)

# Same API as embedded!
with client.session(reversibility=True) as session:

    customer_email = '''
    Hi, this is Robert Chen from Acme Corp.
    My credit card ending in 4532 was charged twice.
    Please refund to robert.chen@acme.com.
    '''

    # Gateway handles all PII detection remotely
    safe_text = session.sanitize(customer_email)

    # Forward to support AI
    response = support_ai.process(safe_text)

    # Restore for human agent
    readable = session.restore(response)
```

### Gateway Server Setup

#### gateway\_server.py

```
# gateway_server.py - Run centrally
from ava.gateway import GatewayServer

server = GatewayServer(
    detection_engine="presidio",
    vault_type="redis",
    vault_config={"host": "redis.company.com", "port": 6379},
    policies_path="/etc/ava/policies/"
)

server.run(host="0.0.0.0", port=8443, tls_cert="/etc/ava/server.crt")
```

## MODE: 3. MOCK ENGINE (Testing/CI)

Regex-based detection with NO dependencies. Perfect for unit tests and CI/CD.

### Unit Testing

#### test\_mock.py

```
import ava
import pytest

@pytest.fixture
def mock_client():
    return ava.Client(engine="mock", policy="general_moderate", vault_type="memory")

def test_email_detection(mock_client):
    with mock_client.session() as session:
        text = "Contact us at support@example.com"
        result = session.sanitize(text)
        assert "AVA_EMAIL_" in result
        assert "support@example.com" not in result

def test_reversibility(mock_client):
    with mock_client.session(reversibility=True) as session:
        original = "Patient: John Doe"
        sanitized = session.sanitize(original)
        restored = session.restore(sanitized)
        assert restored == original
```

NOTE: MockEngine detects: Emails, Phones, SSNs, Credit Cards via regex. No NLP.

## MODE: 4. AWS MACIE ADAPTER

Enterprise-grade PII detection using AWS Macie.

### Prerequisites

#### Install & Configure

```
pip install ava-protocol[aws]
aws configure
```

### AWS Macie Example

#### aws\_macie.py

```
import ava

client = ava.Client(
    engine="aws_macie",
    policy="financial_paranoid",
    vault_type="memory",
    engine_config={
        "region": "us-east-1",
        "custom_data_identifiers": ["employee-id-pattern"]
    }
)

with client.session(reversibility=True) as session:
    # Large document processing
    with open("customer_data.csv", "r") as f:
        content = f.read()

    # AWS Macie detects PII at scale
    sanitized = session.sanitize(content)

    # Process with SageMaker
    insights = sagemaker_model.analyze(sanitized)

    # Restore for reporting
    report = session.restore(insights)
```

## MODE: 5. AZURE PII ADAPTER

Microsoft Azure AI Language PII detection.

### Prerequisites

#### Install

```
pip install ava-protocol[azure]
```

#### Configure

```
export AZURE_LANGUAGE_ENDPOINT=https://your-resource.cognitiveservices.azure.com
export AZURE_LANGUAGE_KEY=your_api_key_here
```

### Azure PII Example

#### azure\_pii.py

```
import ava

client = ava.Client(
    engine="azure_pii",
    policy="healthcare_strict",
    vault_type="redis",
    vault_config={"host": "redis.company.com"},
    engine_config={
        "endpoint": "https://ava-pii.cognitiveservices.azure.com",
        "domain_filter": "phi"
    }
)

with client.session(reversibility=True) as session:
    clinical_notes = '''
    Dr. Sarah Johnson examined patient Michael Brown.
    Patient reports chest pain. Contact: 555-123-4567
    '''

    sanitized = session.sanitize(clinical_notes)

    # Send to Azure OpenAI
    response = azure_openai.ChatCompletion.create(
        deployment_id="gpt-4",
        messages=[{"role": "user", "content": sanitized}]
    )

    final = session.restore(response['choices'][0]['message']['content'])
```

## MODE: 6. GOOGLE DLP ADAPTER

Google Cloud Data Loss Prevention API. 150+ built-in detectors.

### Prerequisites

#### Install

```
pip install ava-protocol[gcp]
gcloud auth application-default login
```

### Google DLP Example

#### google\_dlp.py

```
import ava

client = ava.Client(
    engine="google_dlp",
    policy="legal_confidential",
    vault_type="memory",
    engine_config={
        "project_id": "my-gcp-project",
        "inspect_template": "projects/my-gcp-project/inspectTemplates/legal-template",
        "min_likelihood": "LIKELY"
    }
)

with client.session(reversibility=True) as session:
    legal_document = '''
    ATTORNEY-CLIENT PRIVILEGED
    From: attorney@lawfirm.com
    Re: Merger Discussion
    '''

    # Google DLP detects legal entities
    sanitized = session.sanitize(legal_document)

    # AI summarizes without seeing privileged info
    summary = legal_ai.summarize(sanitized)

    # Restore for attorney review
    privileged_summary = session.restore(summary)
```

# Vault Types (Storage Backends)

## MODE: MEMORY VAULT (Ephemeral)

### memory.py

```
client = ava.Client(
    engine="presidio",
    vault_type="memory" # Default
)
# In-process dict storage
# Data never touches disk
# Auto-purged on session exit
```

NOTE: Best for: Single-session, air-gapped, maximum security.

## MODE: SQLITE VAULT (Persistent)

### sqlite.py

```
client = ava.Client(
    engine="presidio",
    vault_type="sqlite",
    vault_config={
        "db_path": "/secure/ava_vault.db",
        "encryption_key": os.environ["VAULT_KEY"],
        "journal_mode": "WAL"
    }
)
# Survives process restart
# Sessions can be resumed by ID
```

NOTE: Best for: Audit trails, long workflows, recovery.

## MODE: REDIS VAULT (Distributed)

### redis.py

```
client = ava.Client(
    engine="presidio",
    vault_type="redis",
    vault_config={
        "host": "redis.company.com",
        "port": 6379,
        "password": "${REDIS_PASSWORD}",
        "ssl": True
    }
)
# Multiple services share tokens
# Cross-machine session sharing
```

NOTE: Best for: Microservices, load balancers, multi-stage pipelines.

# Policy Configurations

## MODE: BUILT-IN POLICIES

### built\_in.py

```
# HEALTHCARE: HIPAA-compliant
client = ava.Client(engine="presidio", policy="healthcare_strict")
# All 18 PHI identifiers at sensitivity 5
# No retention beyond session

# FINANCIAL: PCI-DSS level 1
client = ava.Client(engine="presidio", policy="financial_paranoid")
# One-time-use tokens for credit cards

# LEGAL: Attorney-client privilege
client = ava.Client(engine="presidio", policy="legal_confidential")
# Extended retention for matter files

# GENERAL: Balanced business use
client = ava.Client(engine="presidio", policy="general_moderate")
# Names/emails protected, dates preserved

# RESEARCH: Scientific data sharing
client = ava.Client(engine="presidio", policy="research_anonymized")
# Irreversible hashing (true anonymization)
```

## MODE: CUSTOM YAML POLICY

### custom.yaml

```
# policies/enterprise_gdpr.yaml
name: enterprise_gdpr
entity_sensitivity:
  PERS: 5 # Always protected
  EMAI: 5
  PHON: 4
  DATE: 2
thresholds:
  min_confidence: 0.85
retention:
  session_ttl: 3600
  audit_retention: 90d
```

### load.py

```
client = ava.Client(engine='presidio', policy='/path/to/custom.yaml')
```

# Async API & Production Workflows

## MODE: ASYNC CLIENT

### async.py

```
import asyncio
import ava

async def process_documents():
    client = ava.AsyncClient(
        engine="presidio",
        policy="general_moderate"
    )

    documents = ["Doc 1...", "Doc 2...", "Doc 3..."]

    async with client.session() as session:
        # Process all concurrently
        tasks = [session.sanitize(doc) for doc in documents]
        sanitized = await asyncio.gather(*tasks)

        # Send to AI concurrently
        ai_tasks = [call_llm(doc) for doc in sanitized]
        responses = await asyncio.gather(*ai_tasks)

        # Restore all concurrently
        restore_tasks = [session.restore(r) for r in responses]
        final = await asyncio.gather(*restore_tasks)

    return final

asyncio.run(process_documents())
```

## MODE: WORKFLOW: Healthcare AI

### healthcare\_api.py

```
import ava
from fastapi import FastAPI

app = FastAPI()
client = ava.Client(engine="presidio", policy="healthcare_strict")

@app.post("/summarize-record")
async def summarize(record_id: str):
    record = ehr_system.get_record(record_id)

    with client.session(reversibility=True, ttl=1800) as session:
        # 1. Sanitize before AI
        safe = session.sanitize(record)

        # 2. Send to OpenAI
        response = openai.ChatCompletion.create(
            model="gpt-4",
            messages=[{"role": "user", "content": safe}]
        )

        # 3. Restore PHI
        summary = session.restore(response['choices'][0]['message']['content'])

        # 4. Audit
        audit_log.store(session.manifest)

    return {"summary": summary}
```