Custom PII Detection Guardrail Implementation Guide

Step-by-step guide for implementing my custom guardrail on litellm and do local development for image building to push to ECR

Technical Assessment Requirements

This guide addresses the specific technical objectives:

1. *Deploy LiteLLM*: with at least one AI model (local container deployment)
2. *Develop custom guardrails*: to detect and block PII(Email + SSN + Credit Card + Phone Number)
3. *Configure LiteLLM*: to use custom guardrails (Regex and Microsoft Presidio)
4. *Understanding Architecture*: with architecture diagram
5. *Demonstrate functionality*: By showing the integrated guardrail into LiteLLM

Prerequisites

- Docker and Docker Compose installed (local development purposes)
- Git installed (version control)
- Basic command line familiarity (enough to work with docker)
- Text editor or IDE (local dev environment for image development)

Step 1: Clone and Setup the Repository

```bash
# Clone the repository
git clone https://github.com/mrcloudchase/litellm-app.git
cd litellm-app

# Verify the structure
ls -la
```

Expected files:
- `Dockerfile` - Container build instructions
- `docker-compose.yml` - Local development stack
- `litellm-config-local.yaml` - LiteLLM configuration used by Docker Compose for local development
- `pii_*.py` - Custom guardrail implementations
- `Makefile` - Automation commands (automates docker compose up/down, etc.)

Step 2: Deploy LiteLLM with Local AI Model

Build and Start the Stack

```bash
# Build all containers
make build

# Start the complete stack (LiteLLM + Ollama + PostgreSQL)
make start

# Pull a local AI model for testing llama3.2:3b
make pull-model

# Verify everything is running
docker ps
```

Expected output:
```
CONTAINER ID   IMAGE                     PORTS
abc123...      litellm-app-litellm      0.0.0.0:4000->4000/tcp
def456...      litellm-app-ollama       0.0.0.0:11434->11434/tcp
ghi789...      postgres:15-apline       0.0.0.0:5432->5432/tcp
```

Verify Base Deployment

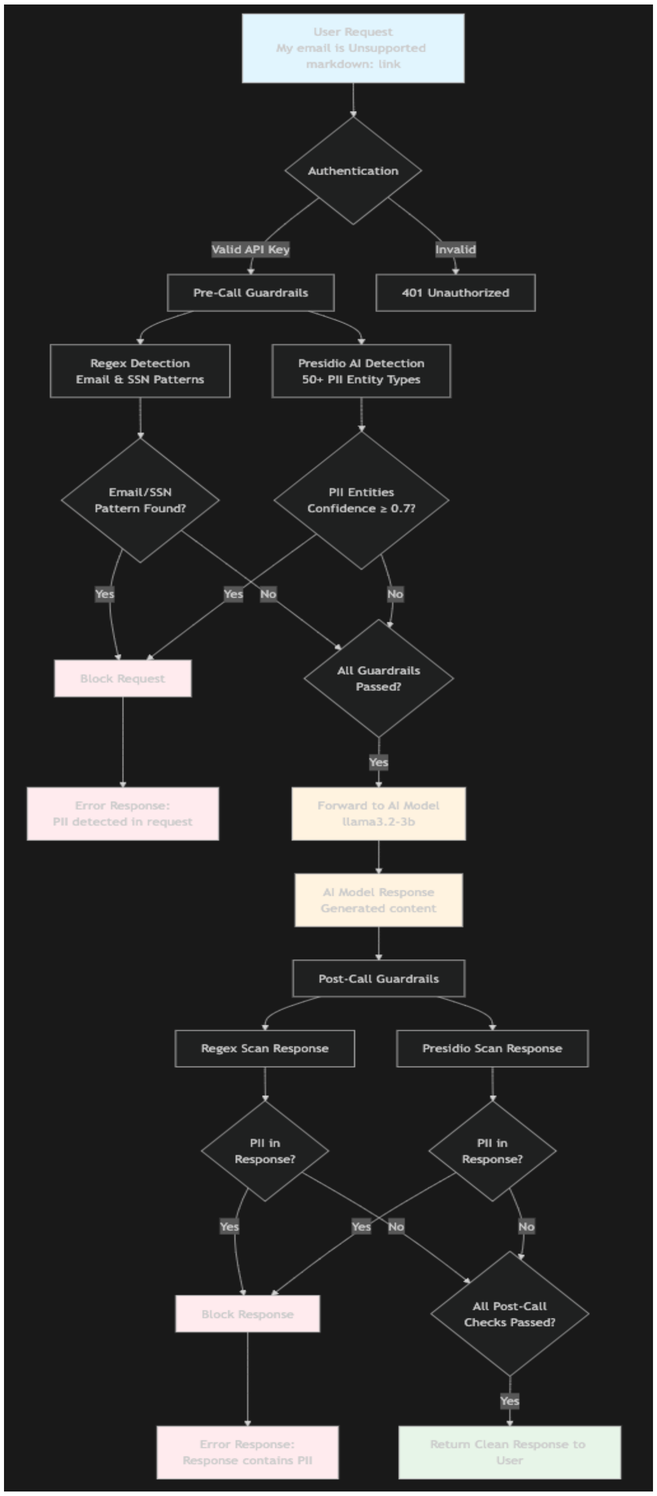
```bash
# Test LiteLLM is responding
curl -X GET http://localhost:4000/health

# Test AI model integration (without guardrails)
curl -X POST http://localhost:4000/v1/chat/completions \
  -H "Authorization: Bearer sk-local-dev-key-12345" \
  -H "Content-Type: application/json" \
  -d '{
    "model": "llama3.2-3b",
    "messages": [{"role": "user", "content": "Hello, how are you?"}],
    "max_tokens": 10
  }'
```

Expected response: JSON with AI-generated content

Step 3: Understanding the Custom Guardrail Implementation

Architecture Overview



Key Implementation Files

1. `pii_regex_detection.py` - Shared regex patterns for email and SSN detection
2. `pii_regex_precall.py` - Pre-call guardrail (blocks PII in user input)
3. `pii_regex_postcall.py` - Post-call guardrail (blocks PII in AI responses)
4. **`pii_presidio_*.py`** - AI-powered guardrails using Microsoft Presidio
5. `litellm-config.yaml` - Configuration that registers the guardrails

Core Guardrail Logic

Email Detection Pattern:
```python
email_pattern = re.compile(r'\b[A-Za-z0-9._%+-]+@[A-Za-z0-9.-]+\.[A-Za-z]{2,}\b')
```

SSN Detection Pattern:
```python
ssn_pattern = re.compile(r'\b\d{3}[-\s]?\d{2}[-\s]?\d{4}\b')
```

Step 4: Test the Custom Guardrails

Test 1: Baseline (No Guardrails)

```bash
# Test without any guardrails - should work normally
curl -X POST http://localhost:4000/v1/chat/completions \
  -H "Authorization: Bearer sk-local-dev-key-12345" \
  -H "Content-Type: application/json" \
  -d '{
    "model": "llama3.2-3b",
    "messages": [{"role": "user", "content": "My email is
test@example.com"}],
    "max_tokens": 10
  }'
```

Expected: Normal AI response (guardrails not active)

Test 2: Email Detection with Regex Guardrail

```bash
# Test email detection - should be BLOCKED
curl -X POST http://localhost:4000/v1/chat/completions \
  -H "Authorization: Bearer sk-local-dev-key-12345" \
  -H "Content-Type: application/json" \
  -d '{
    "model": "llama3.2-3b",
    "messages": [{"role": "user", "content": "My email is
test@example.com"}],
    "guardrails": ["pii-regex-precall"],
    "max_tokens": 10
  }'
```

Expected response:
```json
{
  "error": {
    "message": "Pre-call guardrail blocked PII detected: email",
    "type": "BadRequestError",
    "code": 400
  }
}
```

Test 3: SSN Detection with Regex Guardrail

```bash
# Test SSN detection - should be BLOCKED
curl -X POST http://localhost:4000/v1/chat/completions \
  -H "Authorization: Bearer sk-local-dev-key-12345" \
  -H "Content-Type: application/json" \
  -d '{
    "model": "llama3.2-3b",
    "messages": [{"role": "user", "content": "My SSN is 123-45-6789"}],
    "guardrails": ["pii-regex-precall"],
    "max_tokens": 10
  }'
```

Expected response:
```json
{
  "error": {
    "message": "Pre-call guardrail blocked PII detected: ssn",
    "type": "BadRequestError",
    "code": 400
  }
}
```

Test 4: Clean Input (Should Pass)

```bash
# Test clean input - should work normally
curl -X POST http://localhost:4000/v1/chat/completions \
  -H "Authorization: Bearer sk-local-dev-key-12345" \
  -H "Content-Type: application/json" \
  -d '{
    "model": "llama3.2-3b",
    "messages": [{"role": "user", "content": "Hello, how are you today?"}],
    "guardrails": ["pii-regex-precall"],
    "max_tokens": 10
  }'
```

Expected: Normal AI response (no PII detected)

Test 5: AI-Powered Presidio Guardrail (Stretch Goal)

```bash
# Test comprehensive PII detection with AI
curl -X POST http://localhost:4000/v1/chat/completions \
  -H "Authorization: Bearer sk-local-dev-key-12345" \
  -H "Content-Type: application/json" \
  -d '{
    "model": "llama3.2-3b",
    "messages": [{"role": "user", "content": "My name is John Smith and I
work at Microsoft in Seattle"}],
    "guardrails": ["pii-presidio-precall"],
    "max_tokens": 10
  }'
```

Expected response:
```json
{
  "error": {
    "message": "Pre-call guardrail blocked PII detected: PERSON,
ORGANIZATION, LOCATION",
    "type": "BadRequestError",
    "code": 400
  }
}
```

Step 5: Use HTTP Test Collections (Alternative Testing Method)

Using VS Code REST Client

1. **Install REST Client extension** in VS Code
2. **Open test files:**
   - `tests/test_regex.http` - Regex guardrail tests
   - `tests/test_presidio.http` - Presidio guardrail tests

3. **Run tests interactively:**
   - Click "Send Request" above each test
   - Observe responses in VS Code

Using the Makefile

```bash
# Run automated tests
make test-guardrails

# View test results
cat test_results.log
```

Step 6: Examine the Implementation Code

Regex Detection Engine (`pii_regex_detection.py`)

```python
import re
from typing import Dict, List

class PIIRegexDetection:
    def __init__(self):
        self.patterns = {
            'email': re.compile(r'\b[A-Za-z0-9._%+-]+@[A-Za-z0-9.-]+\.[A-Za-
z]{2,}\b'),
            'ssn': re.compile(r'\b\d{3}[-\s]?\d{2}[-\s]?\d{4}\b'),
            'phone': re.compile(r'\b\(?[0-9]{3}\)?[-.\s]?[0-9]{3}[-.\s]?[0-
9]{4}\b'),
            'credit_card': re.compile(r'\b(?:\d{4}[-\s]?){3}\d{4}\b')
        }

    def detect_pii(self, text: str) -> Dict[str, List[str]]:
        """Detect PII in text and return findings"""
        detected = {}
        for pii_type, pattern in self.patterns.items():
            matches = pattern.findall(text)
            if matches:
                detected[pii_type] = matches
        return detected

    def has_pii(self, text: str) -> bool:
        """Quick check if text contains any PII"""
        return bool(self.detect_pii(text))
```

Pre-Call Guardrail (`pii_regex_precall.py`)

```python
from litellm.integrations.custom_guardrail import CustomGuardrail
from pii_regex_detection import PIIRegexDetection

class PIIRegexPreCallGuardrail(CustomGuardrail):
    def __init__(self, **kwargs):
        self.detector = PIIRegexDetection()
        super().__init__(**kwargs)

    async def async_pre_call_hook(self, user_api_key_dict, cache, data, call_type):
        # Extract user messages
        user_content = ""
        for message in data.get("messages", []):
            if message.get("role") == "user":
                user_content += message.get("content", "") + " "

        # Detect PII
        detected_pii = self.detector.detect_pii(user_content)

        if detected_pii:
            pii_types = list(detected_pii.keys())
            raise Exception(f"Pre-call guardrail blocked PII detected: {', '.join(pii_types)}")

        return None  # Allow request to proceed
```

Configuration Registration (`litellm-config.yaml`)

```yaml
guardrails:
  # Regex-based guardrails (fast, pattern-matching)
  - guardrail_name: "pii-regex-precall"
    litellm_params:
      guardrail: pii_regex_precall.PIIRegexPreCallGuardrail
      mode: "pre_call"

  - guardrail_name: "pii-regex-postcall"
    litellm_params:
      guardrail: pii_regex_postcall.PIIRegexPostCallGuardrail
      mode: "post_call"

  # AI-powered guardrails (comprehensive, context-aware)
  - guardrail_name: "pii-presidio-precall"
    litellm_params:
      guardrail: pii_presidio_precall.PIIPresidioPreCallGuardrail
      mode: "pre_call"
      language: "en"
      threshold: 0.7
```

Step 7: Demonstrate Advanced Features

Multiple Guardrails

```bash
# Use both regex and AI guardrails together
curl -X POST http://localhost:4000/v1/chat/completions \
  -H "Authorization: Bearer sk-local-dev-key-12345" \
  -H "Content-Type: application/json" \
  -d '{
    "model": "llama3.2-3b",
    "messages": [{"role": "user", "content": "Contact me at john@company.com
or call 555-123-4567"}],
    "guardrails": ["pii-regex-precall", "pii-presidio-precall"],
    "max_tokens": 10
  }'
```

Post-Call Protection

```bash
# Test post-call guardrail (if AI model tries to output PII)
curl -X POST http://localhost:4000/v1/chat/completions \
  -H "Authorization: Bearer sk-local-dev-key-12345" \
  -H "Content-Type: application/json" \
  -d '{
    "model": "llama3.2-3b",
    "messages": [{"role": "user", "content": "Generate a fake email address
for testing"}],
    "guardrails": ["pii-regex-postcall"],
    "max_tokens": 50
  }'
```

Step 8: Customize for Your Needs

Adding New PII Types

1. Edit `pii_regex_detection.py`:
```python
self.patterns = {
    'email': re.compile(r'\b[A-Za-z0-9._%+-]+@[A-Za-z0-9.-]+\.[A-Za-z]{2,}\b'),
    'ssn': re.compile(r'\b\d{3}[-\s]?\d{2}[-\s]?\d{4}\b'),
    'phone': re.compile(r'\b\(?[0-9]{3}\)?[-.\s]?[0-9]{3}[-.\s]?[0-9]{4}\b'),
    'credit_card': re.compile(r'\b(?:\d{4}[-\s]?){3}\d{4}\b'),
    # Add your custom pattern here
    'ip_address': re.compile(r'\b(?:[0-9]{1,3}\.){3}[0-9]{1,3}\b'),
}
```

2. Rebuild and test:
```bash
make build
make start
# Test your new pattern
```

Configuring Presidio Entities

1. Edit `pii_presidio_detection.py`:
```python
self.entities = [
    "PERSON", "EMAIL_ADDRESS", "PHONE_NUMBER", "ORGANIZATION",
    "LOCATION", "IP_ADDRESS", "CREDIT_CARD", "US_SSN",
    # Add or remove entities as needed
    "US_PASSPORT", "MEDICAL_LICENSE", "CRYPTO"
]
```

Step 9: Production Deployment Preparation

Container Registry Push

```bash
# Build for production
docker build -t your-registry/litellm-guardrails:latest .

# Push to your registry
docker push your-registry/litellm-guardrails:latest
```

Environment Configuration

```bash
# Set production environment variables
export LITELLM_MASTER_KEY="your-secure-master-key"
export LITELLM_MODE="PRODUCTION"
export DATABASE_URL="postgresql://user:pass@host:5432/litellm"

# Deploy to your infrastructure
docker run -p 4000:4000 \
  -e LITELLM_MASTER_KEY=$LITELLM_MASTER_KEY \
  -e LITELLM_MODE=$LITELLM_MODE \
  -e DATABASE_URL=$DATABASE_URL \
  your-registry/litellm-guardrails:latest
```

Step 10: Walkthrough Preparation

Key Demonstration Points

1. ***Show the running deployment***:
   ```bash
   curl http://localhost:4000/health
   docker ps
   ```

2. ***Demonstrate PII blocking***:
   - Email detection with regex guardrail
   - SSN detection with regex guardrail
   - Person/organization detection with Presidio

3. ***Show clean requests passing through***:
   - Normal conversation without PII
   - AI model responding normally

4. ***Explain the code structure***:
   - Detection engine separation
   - Pre-call vs post-call hooks
   - Configuration in YAML

5. ***Discuss the architecture***:
   - Dual detection strategy (fast + comprehensive)
   - Container-based deployment
   - Production readiness features

Troubleshooting Common Issues

Issue 1: Container Won't Start

```bash
# Check logs
docker logs litellm-app-litellm-1

# Common fix: ensure ports aren't in use
sudo lsof -i :4000
sudo lsof -i :11434
```

Issue 2: Model Not Found

```bash
# Ensure Ollama model is pulled
docker exec litellm-app-ollama-1 ollama list

# Pull the model if missing
make pull-model
```

Issue 3: Guardrail Not Triggering

```bash
# Check configuration
cat litellm-config.yaml

# Verify guardrail is specified in request
# "guardrails": ["pii-regex-precall"]
```

Issue 4: Authentication Errors

```bash
# Verify master key
export MASTER_KEY="sk-local-dev-key-12345"

# Use in requests
curl -H "Authorization: Bearer $MASTER_KEY" ...
```

Success Criteria Checklist

Core Requirements Met:
- [ ] **LiteLLM deployed** with local container and AI model
- [ ] **Custom guardrail developed** with clean, readable code using classes/methods
- [ ] **Two PII types detected**: Email addresses and SSNs
- [ ] **Blocks on input and output**: Pre-call and post-call protection
- [ ] **LiteLLM configured** to use the custom guardrails
- [ ] **Documentation created** with architecture diagram
- [ ] **Functionality demonstrated** with working examples

Stretch Goal Met:
- [ ] **Microsoft Presidio integrated** for comprehensive PII detection
- [ ] **50+ entity types** supported beyond just email/SSN
- [ ] **Context-aware detection** using ML models
- [ ] **Configurable confidence thresholds** for AI detection

Professional Implementation:
- [ ] **Production-ready code** with error handling
- [ ] **Comprehensive testing** with HTTP test collections
- [ ] **Container deployment** with health checks
- [ ] **Configuration management** via YAML
- [ ] **Performance optimization** with dual detection strategy
- [ ] **Documentation** suitable for technical audience

1. **Deployed LiteLLM** with Ollama AI model in containers
2. **Custom PII guardrails** detecting emails, SSNs, and 50+ other entity types
3. **Dual protection** system (regex + AI) for comprehensive coverage
4. **Production-ready** implementation with proper architecture
5. **Comprehensive documentation** with step-by-step instructions
6. **Working demonstration** ready for technical walkthrough

The implementation showcases **enterprise-grade PII protection** with clean, maintainable code that can be easily extended for additional use cases.