skill

1 XAI Grok API Mastery

Complete expertise in XAI's Grok API for building production-ready AI applications with language models, agentic tools, RAG, and multimodal capabilities.

1.1 🎯 Auto-Activation Triggers

This skill **automatically activates** when the user mentions: - “Grok” or “grok” - “XAI” or “x.ai” - “Ask Grok” or “Use Grok” - Model names: “grok-4”, “grok-4-fast”, “grok-3”, etc. - Tools: “web_search”, “x_search” (in XAI context) - Multi-perspective analysis requests - Comparison requests: “Claude vs Grok”, “compare perspectives”

1.2 🚀 Quick Hook Access

Call Grok directly via hooks (use Bash tool):

```
# Non-streaming (default, most queries)
export XAI_API_KEY=$(grep XAI_API_KEY /Users/manu/Documents/LUXOR/
        xai/.env | cut -d '=' -f2) && \
echo '{"prompt": "Your question here", "model": "grok-4-fast"}' |
        ~/.claude/hooks/grok-api.sh

# Streaming (for long responses)
cd /Users/manu/Documents/LUXOR/xai && source venv/bin/activate && \
export XAI_API_KEY=$(grep XAI_API_KEY .env | cut -d '=' -f2) && \
echo '{"prompt": "Your question here", "stream": true}' | python
        ~/.claude/hooks/grok-streaming.py

# With tools (web search)
export XAI_API_KEY=$(grep XAI_API_KEY /Users/manu/Documents/LUXOR/
        xai/.env | cut -d '=' -f2) && \
echo '{"prompt": "Research latest AI", "tools": ["web_search"]}' |
        ~/.claude/hooks/grok-api.sh
```

1.3 Skill Overview

This skill provides comprehensive knowledge of XAI’s Grok API platform, covering all 7 models (5 language, 2 multimodal), 3 server-side agentic tools (web search, X search, code execution), RAG infrastructure via Collections API, and advanced features including reasoning with encrypted thinking traces, massive 2M token context windows, and cost optimization strategies.

1.4 Core Expertise

1.4.1 1. Model Selection & Optimization

- **7 Production Models**: grok-4, grok-4-fast (reasoning/non-reasoning variants), grok-3, grok-3-mini, grok-2-vision, grok-2-image
- **Context Windows**: Up to 2M tokens (largest available)
- **Reasoning Models**: Native reasoning with encrypted thinking traces
- **Cost Optimization**: 98% savings strategies (grok-4-fast vs grok-4)
- **Model Selection Matrix**: Decision trees for optimal model selection by use case

1.4.2 2. Authentication & Security

- **API Key Management**: Creation, rotation, deletion via Console and Management API
- **ACL System**: Fine-grained permissions (endpoints, models, wildcards)
- **Rate Limiting**: QPM, QPS, TPM configuration and monitoring
- **Team Management**: Admin vs member permissions, key ownership
- **Security Best Practices**: Secret management, rotation schedules, least privilege

1.4.3 3. Chat & Completions API

- **Message Structure**: System, user, assistant roles with flexible ordering
- **Streaming**: SSE streaming for real-time responses
- **Deferred Completions**: Long-running tasks with polling
- **Stateful Conversations**: Responses API with previous_response_id
- **Token Management**: Prompt, reasoning, completion token tracking
- **Parameter Tuning**: Temperature, top_p, max_tokens, seed for reproducibility
- **System Fingerprint**: Configuration tracking for debugging

1.4.4 4. Agentic Tools (Server-Side)

- **web_search()**: Real-time internet search with domain filtering
- **x_search()**: X/Twitter search with handle/engagement filtering

- **code_execution()**: Python sandbox for data analysis, calculations, visualization
- **Multi-Tool Orchestration**: Combining tools for complex research workflows
- **Tool Usage Tracking**: Billing and cost monitoring
- **Stateful Agentic Conversations**: Multi-turn tool-augmented dialogs

1.4.5 5. RAG Implementation (Collections API)

- **Collection Management**: Create, list, update, delete collections
- **Document Upload**: HTML, PDF, TXT, Markdown support
- **Chunking Strategies**: Token-based splitting (512-2048 tokens, configurable overlap)
- **Embedding & Indexing**: Automatic with grok-embedding-small
- **Semantic Search**: Multi-collection search with relevance scoring
- **RAG Patterns**: 6 production patterns (basic, iterative, citations, hybrid, conversational, confidence)

1.4.6 6. Image Capabilities

- **Image Understanding**: grok-2-vision-1212 (text + image → text)
  - Input formats: URL, base64 (JPG, PNG up to 20 MiB)
  - Detail parameter: high, low, auto
  - Use cases: OCR, document analysis, visual Q&A
- **Image Generation**: grok-2-image-1212 (text → image)
  - Batch generation: 1-10 images
  - Prompt refinement system
  - Cost: $0.07/image

1.4.7 7. Live Search & Citations

- **Search Modes**: auto, on, off
- **Data Sources**: Web, X, News, RSS feeds
- **Citation Tracking**: Source URLs, snippets, timestamps
- **Source Filtering**: Domain allow/exclude lists, geographic targeting
- **Cost Optimization**: Limiting search results, caching strategies

1.4.8 8. Cost Management

- **Pricing Structure**: Tiered pricing by context window (<128K, >128K)
- **Token Categories**: Prompt (fresh/cached), reasoning, completion
- **Tool Costs**: $2.50 per 100 sources (web/X search)
- **Cost Estimators**: Token-based cost calculation
- **Optimization Strategies**: Model selection, caching, batching, right-sizing

1.4.9 9. SDK Integration

- **Native SDKs**: xai-sdk (Python, TypeScript)
- **Compatible SDKs**: OpenAI SDK, Anthropic SDK (base_url override)
- **Framework Integration**: Vercel AI SDK, LangChain, LlamaIndex
- **gRPC API**: High-performance protocol alternative

1.4.10 10. Production Patterns

- **Error Handling**: HTTP codes, retry logic with exponential backoff
- **Streaming Patterns**: SSE chunking, progressive display
- **Async Workflows**: Deferred completions, background processing
- **Monitoring**: Token usage tracking, cost analysis, performance metrics
- **Debugging**: System fingerprint, encrypted thinking traces, verbose logging

1.5 Use Case Mastery

1.5.1 By Application Type

- **Chatbots**: Model selection, conversation management, memory strategies
- **Research Agents**: Multi-tool orchestration, citation tracking, iterative refinement
- **Document Q&A**: RAG implementation, chunking optimization, context management
- **Image Processing**: OCR, visual analysis, document extraction
- **Content Generation**: Image generation, batch processing, prompt engineering

- **Data Analysis**: Code execution, statistical analysis, visualization

1.5.2 By Industry

- **Finance**: Market research, analysis workflows, compliance
- **Healthcare**: Document processing, research assistance, privacy considerations
- **E-commerce**: Product descriptions, visual search, content generation
- **Customer Support**: Knowledge base Q&A, RAG implementation, escalation
- **Media**: Content generation, image creation, social listening
- **Legal**: Document analysis, OCR, citation tracking
- **Education**: Tutoring systems, Q&A, research assistance

1.6 Technical Capabilities

1.6.1 Code Generation

- Complete Python implementations using xai-sdk
- OpenAI SDK compatibility examples
- curl commands for API testing
- JavaScript/TypeScript examples
- Production-ready error handling
- Async/await patterns
- Streaming implementations
- Batch processing pipelines

1.6.2 Architecture Design

- System architecture diagrams
- Data flow visualization
- Tool orchestration patterns
- State management strategies
- Scaling considerations

- Multi-tenant design
- Caching architectures

1.6.3 Debugging & Troubleshooting

- Common error diagnosis (401, 403, 429, 500)
- Token consumption issues
- Tool calling problems
- Streaming interruptions
- Rate limit debugging
- Performance optimization
- Cost anomaly investigation

1.7 Integration Patterns

1.7.1 Pattern Library

1. **Basic Chat**: Simple conversational interface
2. **Streaming Chat**: Real-time response display
3. **Agentic Research**: Multi-tool autonomous workflows
4. **RAG Knowledge Base**: Document Q&A with semantic search
5. **Image Understanding**: OCR and visual analysis
6. **Image Generation**: Batch content creation
7. **Hybrid RAG**: Collections + Live Search
8. **Conversational RAG**: Stateful document Q&A
9. **Multi-Tool Analysis**: Web + Code execution
10. **Social Listening**: X search with sentiment analysis

1.7.2 SDK Usage Patterns

```
# xAI SDK (Native)
from xai_sdk import Client
from xai_sdk.chat import user, system
from xai_sdk.tools import web_search, code_execution

client = Client(api_key="xai-...")
chat = client.chat.create(model="grok-4-fast", tools=[web_search()])
chat.append(user("Query"))
response = chat.sample()

# OpenAI SDK (Compatible)
from openai import OpenAI
client = OpenAI(api_key="xai-...", base_url="https://api.x.ai/v1")
response = client.chat.completions.create(model="grok-4-fast",
        messages=[...])

# Streaming
for response, chunk in chat.stream():
    print(chunk.content, end="", flush=True)

# RAG
collection = client.collections.create(name="Docs")
client.collections.upload_document(collection_id=..., data=...,
        content_type=...)
results = client.collections.search(query="...", collection_ids=[...])

# Image Understanding
from xai_sdk.chat import image
chat.append(user("Analyze", image(image_url="...", detail="high")))

# Image Generation
images = client.images.generate(model="grok-2-image-1212", prompt="...",
        n=4)
```

1.8 Best Practices

1.8.1 Performance

- Use grok-4-fast for 98% cost savings
- Disable reasoning when not needed (non-reasoning variant)
- Enable prompt caching for repeated prompts (62.5% savings)
- Stay under 128K context to avoid 2× pricing
- Batch multiple queries when possible
- Optimize tool usage (limit search results)
- Pre-process images (compress before upload)

1.8.2 Reliability

- Implement exponential backoff for retries
- Handle rate limiting (429 errors)
- Monitor token consumption
- Use deferred completions for long tasks
- Implement circuit breakers for tools
- Log all tool calls for debugging
- Track system fingerprints for reproducibility

1.8.3 Security

- Rotate API keys every 30-90 days
- Use ACLs for least privilege access
- Never commit keys to version control
- Use secret management systems (AWS Secrets Manager, etc.)
- Separate dev/staging/prod keys
- Monitor usage for anomalies
- Enable audit logging

1.8.4 Cost Optimization

- Right-size model selection (don't always use grok-4)
- Monitor and alert on token spikes
- Implement caching layers
- Use token estimators before requests
- Batch operations where possible
- Optimize chunking for RAG (larger chunks = fewer tokens)
- Set max_tokens limits to prevent runaway costs

1.9 Documentation Reference

1.9.1 Complete Documentation Suite

Located in `/Users/manu/Documents/LUXOR/xai/docs/`:

1. **00-OVERVIEW-AND-CAPABILITIES.md** - Platform overview, architecture, quick reference
2. **01-MODELS-AND-CAPABILITIES.md** - Model catalog, pricing, selection guide (67,000 tokens)
3. **02-AUTHENTICATION-AND-KEYS.md** - API keys, ACLs, rate limits (34 KB)
4. **03-CHAT-AND-COMPLETIONS.md** - Chat API, streaming, deferred (1,500+ lines)
5. **04-AGENTIC-TOOLS.md** - Web search, X search, code execution (8,000 words)
6. **05-COLLECTIONS-AND-RAG.md** - RAG implementation, chunking (64 KB)
7. **06-LIVE-SEARCH-AND-CITATIONS.md** - Search modes, citations (60 KB)
8. **07-IMAGE-CAPABILITIES.md** - Image understanding, generation (95 KB)

1.9.2 Workflow Diagrams

Located in `/Users/manu/Documents/LUXOR/xai/diagrams/workflows.md`: - Authentication flow - Chat completion workflows - Agentic tool orchestration - RAG implementation flow - Image processing workflows - Multi-tool research pipeline - State management patterns - Error handling flow - Cost optimization decision tree

1.10 When to Use This Skill

1.10.1 Automatic Activation

This skill should be automatically activated when: - User mentions “XAI”, “Grok”, “grok-4”, “grok-2-image”, “grok-2-vision” - User asks about XAI API integration - User needs help with Grok model selection - User wants to implement agentic tools (web search, X search, code execution) - User is building RAG systems with XAI - User needs image understanding or generation with Grok - User asks about XAI pricing or cost optimization - User needs authentication/API key management for XAI

1.10.2 Explicit Invocation

User can invoke with: - “Use the XAI Grok API skill” - “Help me with Grok API” - “I need XAI expertise”

1.11 Quick Start Templates

1.11.1 Template 1: Simple Chatbot

```
from xai_sdk import Client
from xai_sdk.chat import user, system

client = Client(api_key="xai-...")
chat = client.chat.create(model="grok-4-fast-non-reasoning")
chat.append(system("You are a helpful assistant."))
chat.append(user("Hello!"))
response = chat.sample()
print(response.content)
```

1.11.2 Template 2: Research Agent

```
from xai_sdk import Client
from xai_sdk.chat import user
from xai_sdk.tools import web_search, code_execution

client = Client(api_key="xai-...")
chat = client.chat.create(
    model="grok-4-fast",
    tools=[web_search(), code_execution()]
)
chat.append(user("Research query"))
for response, chunk in chat.stream():
    print(chunk.content, end="", flush=True)
```

1.11.3 Template 3: RAG System

```
from xai_sdk import Client
from xai_sdk.chat import user

client = Client(api_key="xai-...", management_api_key="xai-...")

# Create collection and upload docs
collection = client.collections.create(name="KB")
# ... upload documents ...

# Query
results = client.collections.search(query="question",
        collection_ids=[collection.collection_id])
context = "\n".join([r.text for r in results.results])

# Generate answer
chat = client.chat.create(model="grok-4-fast")
chat.append(user(f"Context: {context}\n\nQuestion: ..."))
response = chat.sample()
```

1.11.4 Template 4: Image Analysis

```
from xai_sdk import Client
from xai_sdk.chat import user, image

client = Client(api_key="xai-...")
chat = client.chat.create(model="grok-2-vision-1212")
chat.append(user("Analyze", image(image_url="...", detail="high")))
response = chat.sample()
```

1.12 Common Tasks

1.12.1 Task: Choose the Right Model

Decision tree: 1. Need images? → grok-2-vision (understanding) or grok-2-image (generation) 2. Need reasoning? → YES: grok-4-fast-reasoning, NO: grok-4-fast-non-reasoning 3. Budget tight? → grok-4-fast variants (98% savings) 4. Speed critical? → grok-4-fast-non-reasoning (no thinking tokens) 5. High volume? → grok-3-mini (lowest cost language model)

1.12.2 Task: Implement Streaming

```
for response, chunk in chat.stream():
    if response.usage.reasoning_tokens:
        print(f"\rThinking... ({response.usage.reasoning_tokens}
        tokens)", end="")
    if chunk.content:
        print(chunk.content, end="", flush=True)
```

1.12.3 Task: Handle Errors

```
from xai_sdk.exceptions import RateLimitError, APIError
import time

max_retries = 3
for attempt in range(max_retries):
    try:
        response = chat.sample()
        break
    except RateLimitError:
        wait_time = 2 ** attempt
        time.sleep(wait_time)
    except APIError as e:
        if e.status_code >= 500:
            time.sleep(2 ** attempt)
        else:
            raise
```

1.12.4 Task: Estimate Cost

```
def estimate_cost(model, prompt_tokens, completion_tokens):
    prices = {
        "grok-4-fast": (2, 10),
        "grok-4-fast-non-reasoning": (0.20, 1),
        "grok-3-mini": (2, 5),
    }
    if prompt_tokens > 128_000:
        prices = {k: (v[0]*2, v[1]*2) for k, v in prices.items()}

    prompt_price, completion_price = prices[model]
    return (prompt_tokens * prompt_price + completion_tokens *
        completion_price) / 1_000_000

cost = estimate_cost("grok-4-fast", 10_000, 2_000)
print(f"${cost:.4f}")
```

1.13 Skill Maintenance

Last Updated: November 2025 **Source**: Context7 xAI documentation (/websites/x_ai, 245 code snippets) **Coverage**: Complete (100% of public XAI API as of Nov 2025) **Status**: Production-ready

Update Triggers: - New model releases - API endpoint changes - Pricing updates - New tool additions - Breaking changes

1.14 Related Skills

This skill complements: - **API Architecture**: For designing XAI-powered applications - **Python Development**: For SDK usage - **Cloud Architecture**: For deploying XAI applications - **Database Management**: For RAG implementations - **Image Processing**: For multimodal applications

1.15 Success Metrics

Users should be able to: - ✅ Select the optimal XAI model for any use case - ✅ Implement chat completions with streaming in <10 minutes - ✅ Build agentic research tools with web/X search - ✅ Deploy production RAG systems with Collections API - ✅ Process images with understanding and generation - ✅ Optimize costs by 90%+ through strategic model selection - ✅ Debug and troubleshoot XAI API issues independently - ✅ Architect scalable XAI-powered applications

---

Skill Ready: This comprehensive XAI Grok API mastery skill provides production-ready expertise for building any XAI-powered application.