```
import os
import json
import time
import gradio as gr
from datetime import datetime
from typing import List, Dict, Any, Optional, Union
import threading

# Import Groq
from groq import Groq

class CreativeAgenticAI:
    """
    Creative Agentic AI Chat Tool using Groq's compound models
    """

    def __init__(self, api_key: str, model: str = "compound-beta"):
        """
        Initialize the Creative Agentic AI system.

        Args:
            api_key: Groq API key
            model: Which Groq model to use ('compound-beta' or 'compound-beta-mini')
        """
        self.api_key = api_key
        if not self.api_key:
            raise ValueError("No API key provided")

        self.client = Groq(api_key=self.api_key)
        self.model = model
        self.conversation_history = []

    def chat(self, message: str,
             include_domains: List[str] = None,
             exclude_domains: List[str] = None,
             system_prompt: str = None,
             temperature: float = 0.7,
             max_tokens: int = 1024) -> Dict:
        """
        Send a message to the AI and get a response

        Args:
            message: User's message
            include_domains: List of domains to include for web search
            exclude_domains: List of domains to exclude from web search
            system_prompt: Custom system prompt
            temperature: Model temperature (0.0-2.0)
            max_tokens: Maximum tokens in response

        Returns:
            AI response with metadata
        """
        # Default system prompt if none provided
        if not system_prompt:
            system_prompt = """You are a creative and intelligent AI assistant with agentic
capabilities.
            You can search the web, analyze information, and provide comprehensive responses.
            Be helpful, creative, and engaging while maintaining accuracy."""

        # Build messages
        messages = [{"role": "system", "content": system_prompt}]

        # Add conversation history (last 10 exchanges)
        messages.extend(self.conversation_history[-20:])  # Last 10 user-assistant pairs

        # Add current message
        messages.append({"role": "user", "content": message})

        # Set up API parameters
        params = {
            "messages": messages,
            "model": self.model,
            "temperature": temperature,
            "max_tokens": max_tokens
        }

        # Add domain filtering if specified
        if include_domains and include_domains[0].strip():
            params["include_domains"] = [domain.strip() for domain in include_domains if
domain.strip()]
        if exclude_domains and exclude_domains[0].strip():
            params["exclude_domains"] = [domain.strip() for domain in exclude_domains if
domain.strip()]

        try:
            # Make the API call
            response = self.client.chat.completions.create(**params)
            content = response.choices[0].message.content
            # Extract tool usage information
            tool_info = self._extract_tool_info(response)

            # Add to conversation history
            self.conversation_history.append({"role": "user", "content": message})
            self.conversation_history.append({"role": "assistant", "content": content})

            # Create response object
            response_data = {
                "content": content,
                "timestamp": datetime.now().isoformat(),
                "model": self.model,
                "tool_usage": tool_info,
                "parameters": {
                    "temperature": temperature,
                    "max_tokens": max_tokens,
                    "include_domains": include_domains,
                    "exclude_domains": exclude_domains
                }
            }

            return response_data

        except Exception as e:
            error_msg = f"Error: {str(e)}"
            self.conversation_history.append({"role": "user", "content": message})
            self.conversation_history.append({"role": "assistant", "content": error_msg})

            return {
                "content": error_msg,
                "timestamp": datetime.now().isoformat(),
                "model": self.model,
                "tool_usage": None,
                "error": str(e)
            }

    def _extract_tool_info(self, response) -> Dict:
        """Extract tool usage information in a JSON serializable format"""
        tool_info = None
        if hasattr(response.choices[0].message, 'executed_tools'):
            tools = response.choices[0].message.executed_tools
            if tools:
                tool_info = []
                for tool in tools:
                    tool_dict = {
                        "tool_type": getattr(tool, "type", "unknown"),
                        "tool_name": getattr(tool, "name", "unknown"),
                    }
                    if hasattr(tool, "input"):
                        tool_dict["input"] = str(tool.input)
                    if hasattr(tool, "output"):
                        tool_dict["output"] = str(tool.output)
                    tool_info.append(tool_dict)
        return tool_info

    def clear_history(self):
        """Clear conversation history"""
        self.conversation_history = []

    def get_history_summary(self) -> str:
        """Get a summary of conversation history"""
        if not self.conversation_history:
            return "No conversation history"

        user_messages = [msg for msg in self.conversation_history if msg["role"] == "user"]
        assistant_messages = [msg for msg in self.conversation_history if msg["role"] ==
"assistant"]

        return f"Conversation: {len(user_messages)} user messages, {len(assistant_messages)}
assistant responses"

# Global variables
ai_instance = None
api_key_status = "Not Set"

def validate_api_key(api_key: str, model: str) -> str:
    """Validate Groq API key and initialize AI instance"""
    global ai_instance, api_key_status

    if not api_key or len(api_key.strip()) < 10:
        api_key_status = "Invalid ❌"
        return "❌ Please enter a valid API key (should be longer than 10 characters)"

    try:
        # Test the API key
        client = Groq(api_key=api_key)
        # Try a simple request to validate
        test_response = client.chat.completions.create(
            messages=[{"role": "user", "content": "Hello"}],
            model=model,
            max_tokens=10
        )

        # Create AI instance
        ai_instance = CreativeAgenticAI(api_key=api_key, model=model)
        api_key_status = "Valid ✅"

        return f"✅ API Key Valid! Creative Agentic AI is ready.\n\n**Model:** {model}\n**Status:**
Connected and ready for chat!"

    except Exception as e:
        api_key_status = "Invalid ❌"
        ai_instance = None
        return f"❌ Error validating API key: {str(e)}\n\nPlease check your API key and try again."

def update_model(model: str) -> str:
    """Update the model selection"""
    global ai_instance

    if ai_instance:
        ai_instance.model = model
        return f"✅ Model updated to: **{model}**"
    else:
        return "⚠️ Please set your API key first"

def chat_with_ai(message: str,
                include_domains: str,
                exclude_domains: str,
                system_prompt: str,
                temperature: float,
                max_tokens: int,
                history: List) -> tuple:
    """Main chat function"""
    global ai_instance

    if not ai_instance:
        error_msg = "⚠️ Please set your Groq API key first!"
        history.append([message, error_msg])
        return history, ""

    if not message.strip():
        return history, ""

    # Process domain lists
    include_list = [d.strip() for d in include_domains.split(",")] if include_domains.strip() else []
    exclude_list = [d.strip() for d in exclude_domains.split(",")] if exclude_domains.strip() else []

    try:
        # Get AI response
        response = ai_instance.chat(
            message=message,
            include_domains=include_list if include_list else None,
            exclude_domains=exclude_list if exclude_list else None,
            system_prompt=system_prompt if system_prompt.strip() else None,
            temperature=temperature,
            max_tokens=int(max_tokens)
        )

        # Format response
        ai_response = response["content"]

        # Add tool usage info if available
        if response.get("tool_usage"):
            ai_response += f"\n\n*🔧 Tools used: {len(response['tool_usage'])} tool(s)*"

        # Add to history
        history.append([message, ai_response])

        return history, ""

    except Exception as e:
        error_msg = f"❌ Error: {str(e)}"
        history.append([message, error_msg])
        return history, ""

def clear_chat_history():
    """Clear the chat history"""
    global ai_instance
    if ai_instance:
        ai_instance.clear_history()
    return []

def create_gradio_app():
    """Create the main Gradio application"""
    # Custom CSS for better styling
    css = """
    .container {
        max-width: 1200px;
        margin: 0 auto;
    }
    .header {
        text-align: center;
        background: linear-gradient(to right, #00ff94, #00b4db);
        color: white;
        padding: 20px;
        border-radius: 10px;
        margin-bottom: 20px;
    }
    .status-box {
        background-color: #f8f9fa;
        border: 1px solid #dee2e6;
        border-radius: 8px;
        padding: 15px;
        margin: 10px 0;
    }
    #neuroscope-accordion {
        background: linear-gradient(to right, #00ff94, #00b4db);
        border-radius: 8px;
    }
    """

    with gr.Blocks(css=css, title="🤖 Creative Agentic AI Chat", theme=gr.themes.Ocean()) as
app:

        # Header
        gr.HTML("""
        <div class="header">
            <h1>🤖 NeuroScope-AI</h1>
            <p>Powered by Groq's Compound Models with Web Search & Agentic Capabilities</p>
        </div>
        """)

        # API Key Section
        with gr.Row():
            api_key = gr.Textbox(
                label="🔑 Groq API Key",
                placeholder="Enter your Groq API key here...",
                type="password",
                info="Get your API key from: https://console.groq.com/"
            )
            model_selection = gr.Radio(
                choices=["compound-beta", "compound-beta-mini"],
                label="🧠 Model Selection",
                value="compound-beta",
                info="compound-beta: More powerful | compound-beta-mini: Faster"
            )
            connect_btn = gr.Button("🔗 Connect", variant="primary", size="lg")

        # Status display
        status_display = gr.Markdown("### 📊 Status: Not connected",
elem_classes=["status-box"])

        # Connect button functionality
        connect_btn.click(
            fn=validate_api_key,
            inputs=[api_key, model_selection],
            outputs=[status_display]
        )

        model_selection.change(
            fn=update_model,
            inputs=[model_selection],
            outputs=[status_display]
        )

        # Main Chat Interface
        with gr.Tab("💬 Chat"):
            chatbot = gr.Chatbot(
                label="Creative AI Assistant",
                height=500,
                show_label=True,
                bubble_full_width=False,
                show_copy_button=True
            )

            with gr.Row():
                msg = gr.Textbox(
                    label="Your Message",
                    placeholder="Type your message here...",
                    lines=3
                )
                with gr.Column():
```

```
            send_btn = gr.Button("📤 Send", variant="primary")
            clear_btn = gr.Button("🗑️ Clear", variant="secondary")

    # Advanced Settings
    with gr.Accordion("⚙️ Advanced Settings", open=False, elem_id="neuroscope-accordion"):
        with gr.Row():
            temperature = gr.Slider(
                minimum=0.0,
                maximum=2.0,
                value=0.7,
                step=0.1,
                label="🌡️ Temperature",
                info="Higher = more creative, Lower = more focused"
            )
            max_tokens = gr.Slider(
                minimum=100,
                maximum=4000,
                value=1024,
                step=100,
                label="📝 Max Tokens",
                info="Maximum length of response"
            )

        system_prompt = gr.Textbox(
            label="🎭 Custom System Prompt",
            placeholder="Override the default system prompt...",
            lines=2,
            info="Leave empty to use default creative assistant prompt"
        )

    # Domain Filtering Section
    with gr.Accordion("🌐 Domain Filtering (for Web Search)", open=False,
elem_id="neuroscope-accordion"):
        with gr.Row():
            include_domains = gr.Textbox(
                label="✅ Include Domains (comma-separated)",
                placeholder="arxiv.org, *.edu, github.com, stackoverflow.com",
                info="Only search these domains"
            )
            exclude_domains = gr.Textbox(
                label="❌ Exclude Domains (comma-separated)",
                placeholder="wikipedia.org, reddit.com, twitter.com",
                info="Never search these domains"
            )
    # Event handlers
    send_btn.click(
        fn=chat_with_ai,
        inputs=[msg, include_domains, exclude_domains, system_prompt, temperature,
max_tokens, chatbot],
        outputs=[chatbot, msg]
    )

    msg.submit(
        fn=chat_with_ai,
        inputs=[msg, include_domains, exclude_domains, system_prompt, temperature,
max_tokens, chatbot],
        outputs=[chatbot, msg]
    )

    clear_btn.click(
        fn=clear_chat_history,
        outputs=[chatbot]
    )
return app

# Main execution
if __name__ == "__main__":
app = create_gradio_app()
app.launch(
    share=True
)
```