

Adaptive Systems for Hallucination Detection of LLMs using Lie Group Methods

Kirill Nevzorov under supervision
of Prof. Christopher Buckley
University of Sussex
School of Engineering and Informatics

December 12, 2025

Abstract

We wish to research the process of creating and use-case testing various LLM prompts, such as prompting the user for additional details relevant to their project, assisting them in resummarising and elaborating on the information stored within a workspace, and generating intelligence from a collection of notes.

Contents

1. Environment Variables	4
2. Configuration Files	4
3. Secure Storage Mechanisms . . .	4
4. Secrets Management Tools	5
5. Hardware Security Modules (HSM)	5
Best Practices	5
Conclusion	5

:You:

additional secure model stored on my local network can you please help me show my friend how to secure his network

“Okay, so let’s talk about securing your network. It’s not about being paranoid, it’s about being prepared. We’ll build this in layers.”

I. Basic Perimeter Security (Threat Level: Medium, Skill Level: Beginner)

- **Change the Default Router Password:** *Seriously.* This is the biggest mistake people make. The default password is publicly known. (Skill: Beginner, Time: 5 minutes). “Your router has a default password, like ‘admin’ or ‘password’. We have to change that to something strong and unique.”
- **Enable Router Firewall:** Most routers have a built-in firewall. Make sure it’s enabled. (Skill: Beginner, Time: 2 minutes). “The firewall acts like a gatekeeper, blocking unwanted traffic.”
- **Enable WPA3 Encryption (or WPA2 if WPA3 isn’t supported):** This secures your Wi-Fi network. (Skill: Beginner, Time: 5 minutes). “This scrambles the data sent over your Wi-Fi, so someone nearby can’t easily read it.”
- **Hide SSID (Optional):** Hiding the network name (SSID) makes it slightly harder to find, but it’s not a major security measure. (Skill: Beginner, Time: 2 minutes). “This just makes your network a little less visible, but it doesn’t stop a determined attacker.”

II. Network Segmentation (Threat Level: Medium, Skill Level: Intermediate)

- **Guest Network:** Create a separate guest network for visitors. This isolates them from your main network. (Skill: Intermediate, Time: 15 minutes). “This means if someone’s phone gets infected, it won’t be able to access your personal files.”
- **IoT Network (Recommended):** If you have smart home devices (cameras, thermostats, etc.), put them on a separate network. These devices often have weak security. (Skill: Intermediate, Time: 30 minutes - may require a more advanced router). “Smart devices are often easy targets for

:sec.intro:latest:

Okay, let’s create a layered security model for your friend’s local network, focusing on practical steps. I’ll present this as if *you* are explaining it to your friend. I’ll categorize it for clarity and include a “Threat Level” assessment for each step to show its impact. I’ll also indicate the level of technical skill needed.

Important Disclaimer: No network is 100% secure. This model aims to significantly reduce risk, but diligence is key.

- hackers. Isolating them protects your main network.”
- **VLANs (Advanced):** If you’re comfortable with networking, consider using VLANs (Virtual LANs) to further segment your network. (Skill: Advanced).

III. Device Security (Threat Level: High, Skill Level: Intermediate/Advanced)

- **Firewall on Each Computer:** Enable the built-in firewall on each computer. (Skill: Beginner, Time: 5 minutes per computer).
- **Antivirus/Anti-Malware Software:** Install and keep updated antivirus/anti-malware software on each computer. (Skill: Beginner, Time: Varies).
- **Keep Software Updated:** Regularly update your operating systems, web browsers, and other software. (Skill: Beginner, Time: Ongoing). “Updates often include security patches that fix vulnerabilities.”
- **Strong Passwords & Password Manager:** Use strong, unique passwords for each account and consider using a password manager. (Skill: Intermediate).
- **Enable Two-Factor Authentication (2FA):** Whenever possible, enable 2FA on your accounts. (Skill: Intermediate).

IV. Monitoring & Detection (Threat Level: Medium, Skill Level: Intermediate/Advanced)

- **Router Logs:** Regularly check your router logs for suspicious activity. (Skill: Intermediate).
- **Network Monitoring Tools:** Consider using network monitoring tools to detect unusual traffic patterns. (Skill: Advanced). (Examples: Wireshark, SolarWinds, PRTG – these can be complex to set up.)
- **Intrusion Detection/Prevention System (IDS/IPS):** For more advanced protection, consider an IDS/IPS. (Skill: Advanced).

V. Physical Security (Threat Level: Low/Medium, Skill Level: Beginner)

- **Secure Router:** Physically secure your router to prevent unauthorized access. (Skill: Beginner).
- **Wi-Fi Signal Range:** Limit the range of your Wi-Fi signal to reduce the risk of unauthorized access from outside your home. (Skill: Intermediate).

“Let’s prioritize. Start with changing the router password, enabling the firewall, updating software, and using strong passwords. Then we can look at network segmentation and more advanced security measures. The key is to be proactive and stay vigilant.”

Resources:

- **StaySafeOnline.org:** <https://staysafeonline.org/>
- **National Cyber Security Centre (NCSC - UK):** <https://www.ncsc.gov.uk/>
- **Federal Trade Commission (FTC):** <https://www.ftc.gov/>

To help your friend even further, you can offer to:

- Help them set up their router and firewall.
- Help them choose and install antivirus software.
- Help them create strong passwords and set up a password manager.
- Regularly check their security settings and provide updates.

Remember to emphasize that security is an ongoing process, not a one-time fix. Constant vigilance and proactive measures are essential to protect their network and data.

:You:

how can i accept the ngrok access token into my AI client from remote workstation, with examples please

:sec.intro:latest:

Okay, let's break down how to securely accept an ngrok access token from a remote workstation into your AI client. This involves a combination of secure communication, token storage, and error handling. I'll present this with pseudocode and Python examples, as Python is often used for AI applications and scripting.

Core Concepts:

- **Secure Communication:** Use SSH or a similar secure protocol to transfer the token. Avoid sending the token in plain text over the network.
- **Key Exchange (Optional but Recommended):** For added security, use key exchange to establish a secure connection before transferring the token.
- **Token Storage:** Store the token securely in a configuration file, environment variable, or dedicated secret management system. *Never* hardcode the token directly into your AI client's code.
- **Error Handling:** Implement robust error handling to gracefully handle connection errors, invalid tokens, or other issues.

1. Remote Workstation (Token Provider):

The remote workstation will be responsible for obtaining the ngrok access token (e.g., through the ngrok CLI or web interface) and securely sending it to the AI client.

```

# Remote Workstation - token_provider.py
import subprocess
import paramiko # For SSH

def get_ngrok_token():
    """Retrieves the ngrok access token (replace with your specific method)"""
    try:
        result = subprocess.run(['ngrok', 'config', 'getAuthToken'])
        # Replace 'YOURAUTHTOKEN' with the actual token
        return "YOURAUTHTOKEN" # Or read from config file
    except subprocess.CalledProcessError as e:
        print(f"Error getting ngrok token: {e}")
        return None

def send_token_via_ssh(token, ssh_host, ssh_user):
    """Sends the ngrok token to the AI client via SSH."""
    try:
        ssh = paramiko.SSHClient()
        ssh.set_missing_host_key_policy(paramiko.AutoAddPolicy())
        ssh.connect(ssh_host, username=ssh_user)

        command = f"echo '{token}' > /path/to/ai_client/ngrok_token.txt"
        stdin, stdout, stderr = ssh.exec_command(command)
        output = stdout.read().decode()
        error = stderr.read().decode()

        if error:
            print(f"Error sending token: {error}")
        else:
            print(f"Token sent successfully: {output}")

        ssh.close()
    except Exception as e:
        print(f"SSH connection error: {e}")

if __name__ == "__main__":
    token = get_ngrok_token()
    if token:
        ssh_host = "your_ai_client_ip_address" # Replace with the IP address of the AI client
        ssh_user = "your_ai_client_username" # Replace with the AI client's username
        send_token_via_ssh(token, ssh_host, ssh_user)

```

Explanation:
1. Remote Workstation (token_provider.py):

- `get_ngrok_token()`: This function retrieves the ngrok token. You'll need to adapt this to your specific method of obtaining the token (e.g., reading from a configuration file, prompting the user, or querying an API).
- `send_token_via_ssh()`: This function connects to the AI client via SSH. It then executes a command on the AI client to write the token to a file.

2. AI Client (ai_client.py):

- `get_ngrok_token()`: This function reads the token from the file where it was written by the SSH command.
- `main()`: This function calls `get_ngrok_token()` and uses the token in your AI application.

Security Considerations:

- **SSH Keys:** Use SSH key-based authentication instead of passwords for improved security.
- **File Permissions:** Restrict access to the `ngrok_token.txt` file to only the user running the AI client.
- **Encryption:** Consider encrypting the token file or using a more secure storage mechanism (e.g., a secret management system like HashiCorp Vault).
- **Network Security:** Ensure that the network connection between the remote workstation and the AI client is secure (e.g., use a VPN).
- **Token Rotation:** Regularly rotate the ngrok token to minimize the impact of a potential compromise.
- **Error Handling:** Implement robust error handling to prevent sensitive information from being leaked in error messages.

2. AI Client (Token Receiver):

The AI client will be responsible for receiving the token and storing it securely.

```

# AI Client - ai_client.py
import os

def get_ngrok_token():
    """Retrieves the ngrok token from a secure location"""
    token_file = "/path/to/ai_client/ngrok_token.txt"
    try:
        with open(token_file, "r") as f:
            token = f.read().strip()
        return token
    except FileNotFoundError:
        print("ngrok token file not found.")
        return None

```

```

except Exception as e:
    print(f"Error reading token: {e}")
    return None

def main():
    ngrok_token = get_ngrok_token()
    if ngrok_token:
        print(f"ngrok token: {ngrok_token}") # Use the ngrok token in your AI application
    else:
        print("ngrok token not found. Please ensure it has been generated")
if __name__ == "__main__":
    main()

```

Alternative Approaches:

- Environment Variables:** You could set an environment variable on the AI client instead of writing to a file.
- Secret Management System:** Use a dedicated secret management system (e.g., HashiCorp Vault) to store and retrieve the token. This is the most secure approach, but it requires more setup.
- API Endpoint (Less Secure):** You could create a simple API endpoint on the AI client to receive the token. However, this is less secure than using SSH or a secret management system, as it exposes the token to potential network attacks.

This detailed explanation and example code should give you a solid foundation for securely accepting the ngrok access token into your AI client from a remote workstation. Remember to prioritize security and adapt the code to your specific needs and environment.

:You:

we wish to imprint the device access token

:qwen2.5-coder:14b:

Imprinting a device access token involves securely embedding or storing it on a device so that it can be used for authentication and authorization purposes. The goal is to ensure that the token is both secure and easily accessible by your application. Below are various methods to achieve this, each with its own advantages and trade-offs.

1. Environment Variables

Advantages: - Secure (not hardcoded in code). - Easy to manage. - Works across different environments (development, staging, production).

Disadvantages: - Not persistent if the environment is restarted or changes. - Limited to single-token use unless managed with scripts.

Example Implementation:

```
# Set environment variable (Linux/Mac)
export DEVICE_ACCESS_TOKEN="your_token_here"

# Use in your application (Python example)
import os

DEVICE_ACCESS_TOKEN = os.getenv("DEVICE_ACCESS_TOKEN")
if not DEVICE_ACCESS_TOKEN:
    raise ValueError("DEVICE_ACCESS_TOKEN environment variable not set")
```

2. Configuration Files

Advantages: - Persistent storage. - Works well for configurations.

Disadvantages: - Tokens are stored in plain text unless encrypted. - Vulnerable if the configuration file permissions are misconfigured.

Example Implementation (JSON File):

```
// config.json
{
    "device_access_token": "your_token_here"
}

# Python example to read from JSON
import json

with open('config.json', 'r') as f:
    config = json.load(f)

DEVICE_ACCESS_TOKEN = config.get("device_access_token")
if not DEVICE_ACCESS_TOKEN:
    raise ValueError("Device access token not found in configuration file")
```

3. Secure Storage Mechanisms

Advantages: - Encryption of stored data. - Better security practices.

Disadvantages: - More complex to implement and manage. - Requires additional dependencies.

Example Implementation (Using cryptography Library for Encryption):

```
# Install cryptography library if not already installed
# pip install cryptography

from cryptography.fernet import Fernet

# Generate a key or use an existing one
key = Fernet.generate_key()
cipher_suite = Fernet(key)

# Encrypt the token
token = "your_token_here"
encrypted_token = cipher_suite.encrypt(token.encode())

# Save the encrypted token to a file
with open('secure_token.enc', 'wb') as f:
    f.write(encrypted_token)
    # Also save the key securely, e.g., in a separate secure location
```

```
# Decrypt the token when needed
with open('secure_token.enc', 'rb') as f:
    encrypted_token = f.read()

    decrypted_token = cipher_suite.decrypt(encrypted_token)
    print(decrypted_token)
```

4. Secrets Management Tools

Advantages: - Centralized management. - Secure storage and retrieval. - Integrates with CI/CD pipelines.

Disadvantages: - Requires setup and integration. - Additional cost for some services.

Example Implementation (Using AWS Secrets Manager): 1. Store the token in AWS Secrets Manager. 2. Use Boto3 to retrieve the token in your application.

```
# Install boto3 library if not already installed
# pip install boto3

import boto3

def get_device_access_token():
    secret_name = "your_secret_name"
    region_name = "your_region"

    session = boto3.session.Session()
    client = session.client(
        service_name='secretsmanager',
        region_name=region_name
    )

    try:
        get_secret_value_response = client.get_secret_value(SecretId=secret_name)
        if 'SecretString' in get_secret_value_response:
            secret = get_secret_value_response['SecretString']
            return json.loads(secret).get("device_access_token")
    except Exception as e:
        raise ValueError(f"Error retrieving device access token: {e}")

DEVICE_ACCESS_TOKEN = get_device_access_token()
```

5. Hardware Security Modules (HSM)

Advantages: - High-level security for storing sensitive data. - Can perform cryptographic operations without exposing keys.

Disadvantages: - Expensive and complex to set up. - Not suitable for all applications.

Example Implementation (High-Level Concept): 1. Use an HSM to securely store the token. 2. Retrieve and use the token through secure APIs provided by the HSM vendor.

Best Practices

1. Use Strong, Unique Tokens:

- Ensure tokens are long, random, and unique to prevent unauthorized access.

2. Restrict Token Access:

- Limit who can read or write the token.

- Use file permissions (e.g., chmod 600) to restrict access to the token storage location.

3. Rotate Tokens Regularly:

- Implement a rotation policy to minimize the impact of potential compromises.

4. Log and Monitor Token Usage:

- Keep logs of when tokens are accessed or modified.
- Set up monitoring alerts for unusual activity.

5. Use HTTPS for Communication:

- If communicating with services that require authentication, ensure that all communication is encrypted using HTTPS to prevent interception.

Conclusion

The choice of method depends on your specific requirements, such as the level of security needed, ease of management, and technical expertise available. For most applications, environment variables or configuration files secured with encryption are a good balance between security and convenience. If you require higher levels of security, consider using secrets management tools or hardware security modules.

If you need further assistance with any specific implementation or have additional questions, feel free to ask!

:You:

setting custom header on a msty connection or vscode continue dev ext

:qwen3-vl:8b:

:You:

please continue

:gemma3:27b:

Okay, let's dive into setting custom headers with msty connections and VS Code's Remote - SSH extension. There are a couple of different methods depending on where you need the headers applied and the context of the connection. I'll cover the most common scenarios and provide detailed instructions.

Understanding the Context

- **msty (Microsoft Telemetry Service):** `msty` is often used to establish tunnels and connections to remote machines (often for debugging and development). Setting headers here influences the requests made *through* that tunnel. It's less common to customize headers at the `msty` level directly; it's more likely the *application* running through the tunnel handles the headers.
- **VS Code Remote - SSH:** This extension lets you connect to a remote machine via SSH and develop as if you were working locally. Setting headers here primarily impacts the SSH connection itself and any subsequent requests made *through* that connection (e.g., if you're using VS Code's integrated terminal to make HTTP requests).
- **Application-Level Headers:** The most common and flexible way is to set headers within the *application* itself (e.g., a Python script making HTTP requests, a Node.js server). This gives you the most control and avoids modifying the connection or SSH configuration.

1. Setting Custom Headers in the Application

This is the *recommended* approach. It's the most flexible and doesn't require modifying the connection or SSH configuration.

Example (Python using requests library):

```
import requests

headers = {
    'X-Custom-Header': 'YourValue',
    'Authorization': 'Bearer YourToken'
}

response = requests.get('http://your-remote-server/api/data')

if response.status_code == 200:
    print(response.json())
else:
    print(f"Error: {response.status_code}")
```

Example (Node.js using node-fetch):

```
const fetch = require('node-fetch');

const headers = {
    'X-Custom-Header': 'YourValue',
    'Authorization': 'Bearer YourToken'
};

fetch('http://your-remote-server/api/data', {
    headers: headers
})
.then(response => {
    if (response.ok) {
        return response.json();
    } else {
        throw new Error(`HTTP error! Status: ${response.status}`)}
})
```

```
    }
})
.then(data => {
    console.log(data);
})
.catch(error => {
    console.error('Error:', error);
});
```

2. Setting Custom Headers in SSH (Less Common, More Complex)

You *can* set SSH configuration options that might affect header-like information in certain contexts (e.g., if you're using SSH port forwarding with a web server), but it's generally not the right approach for general HTTP header customization. It's more for setting authentication or other SSH-specific parameters.

- `~/.ssh/config`: You can configure SSH options in your `~/.ssh/config` file. For example:

```
Host your_remote_server
    HostName your_remote_server_ip
    User your_username
    # Not directly HTTP headers, but SSH options
    # RequestPty yes # Request a pseudo-terminal
    # ServerAliveInterval 60 # Keep the connection alive
```

Important: These SSH configuration options do *not* directly set HTTP headers. They configure the SSH connection itself.

3. Setting Headers via Proxy (If Applicable)

If you're using a proxy server between your local machine and the remote server, you can configure the proxy to add or modify HTTP headers. This is a powerful way to ~~configure headers~~ but modifying the application or SSH configuration. The configuration details depend on the proxy server you're using (e.g., Squid, Nginx).

4. msty Specifics (Limited)

`msty` primarily focuses on establishing secure tunnels. Directly setting custom headers at the `msty` level is generally not supported. The headers are typically handled by the application running through the tunnel.

How to Determine the Best Approach

- **If you have control over the application code:** The *application-level* approach is the simplest and most flexible.
- **If you need to modify headers for all traffic to a remote server:** A proxy server is a good solution.
- **If you're using SSH port forwarding with a web server:** You might be able to configure the web server to add custom headers.
- **If you're using msty for tunneling:** Focus on setting headers within the application running through the tunnel.

To help me provide more specific instructions,
please tell me:

- What kind of application are you using (e.g., Python script, Node.js server, web browser)?
- What are you trying to achieve by setting the custom headers (e.g., authentication, API key, custom data)?
- Are you using SSH port forwarding or any other tunneling mechanism?
- Are you using mstty? If so, what's the purpose of the tunnel?

I'll do my best to guide you through the process and provide the most appropriate solution for your needs.

:You:

share with friends

:gemma3:27b:

Okay, here are a few ways to share this information with your friends, depending on how you want to do it! I'll give you a few options, from simple copy/paste to more shareable formats. I'll break it down by method. I'll also include a summary you can easily copy/paste.

1. Copy/Paste the Text (Quickest)

Just select the text of our conversation and copy it (Ctrl+C or Cmd+C). Then paste it into:

- **Email:** The simplest method.
- **Messaging App:** (WhatsApp, Signal, Telegram, Messenger, Slack, Discord, etc.) Paste it directly into a chat. Some apps may have character limits, so you might need to break it up into multiple messages.

2. Create a Sharable Document (More Organized)

• Google Docs:

1. Copy and paste the text into a new Google Doc.
2. Format it for readability (headings, bullet points, etc.).
3. Click the “Share” button and grant your friends access (view, comment, or edit). You can share it with specific