

# Anleitung zur Entwicklung und Implementierung von Small Language Models (SLMs) mit Ollama für Cybersecurity-Anwendungen

## I. Grundlagen

### 1. Einführung: Warum SLMs für Cybersecurity?

Small Language Models (SLMs) sind kompakte KI-Modelle, die speziell für Aufgaben mit begrenztem Kontext entwickelt wurden. Im Vergleich zu großen Sprachmodellen (LLMs) zeichnen sie sich durch:

- **geringen Ressourcenverbrauch** (CPU-fähig, lokal einsetzbar),
- **schnelle Inferenzzeiten** (niedrige Latenz, auch bei Serienauswertung),
- **datenschutzfreundlichen Einsatz** (Verarbeitung sensibler Daten ohne Cloud),
- **einfaches Prompt-Tuning ohne Feintuning-Infrastruktur** aus.

Für den Einsatz im Bereich der Cybersicherheit eignen sich SLMs insbesondere dort, wo:

- **Textdaten analysiert oder klassifiziert** werden müssen (z. B. E-Mails, Logzeilen, CVEs),
- **kontextsensitive Entscheidungen** erforderlich sind (z. B. SOC-Ticket-Triage),
- **kein Cloud-Zugriff erlaubt** oder gewünscht ist.

Gerade in kleinen IT-Teams oder KMU, die keine umfassenden SIEM- oder XDR-Lösungen betreiben, ermöglichen SLMs eine neue Form der intelligenten Assistenz bei sicherheitsrelevanten Routineaufgaben.

### 2. Typische Anwendungsbereiche

Folgende Use Cases sind besonders gut geeignet für SLM-basierte Analysen:

- **Phishing-Erkennung und -Bewertung** in E-Mails oder Supportanfragen
- **Analyse von Logdateien** (z. B. verdächtige Login-Vorgänge, Fehlercodes)
- **Triage von Tickets oder Incidents** (z. B. SOC- oder Helpdesk-Umgebungen)
- **Klassifikation und Priorisierung von CVEs** (Common Vulnerabilities and Exposures)
- **Extraktion sicherheitsrelevanter Informationen** aus unstrukturiertem Text
- **Unterstützung bei der Dokumentation** sicherheitsrelevanter Prozesse

Diese Szenarien zeichnen sich durch klare Eingaben, domänenspezifischen Kontext und hohe Wiederholbarkeit aus – ideale Bedingungen für den Einsatz kleiner Sprachmodelle.

## II. Voraussetzungen

### 1. Python-Umgebung und Ollama als technische Basis

Die Entwicklung und Integration eines Small Language Models (SLM) in eine Cybersecurity-Anwendung setzt eine geeignete Entwicklungsumgebung voraus. Zwei zentrale Bestandteile sind dabei:

#### a) Python ab Version 3.10

Python ist die Grundlage für die Scripting-Logik und API-Integration. Es wird verwendet, um:

- Prompts und Aufgaben automatisiert an das Modell zu übermitteln,
- Ausgaben zu analysieren und weiterzuverarbeiten (z. B. in einem Filter- oder Analyse-Skript),
- das SLM in bestehende Sicherheitslösungen (z. B. Mailserver, SIEM, Logparser) einzubetten.

#### Empfohlen:

- Verwende Python  $\geq 3.10$
- Installiere optimal miniconda/anaconda oder eine gleichwertige virtuelle Umgebung

### 2. Modellbereitstellung und -verwendung mit Ollama

Nach erfolgreicher Installation kann Ollama genutzt werden, um passende SLMs zu laden und lokal bereitzustellen.

#### a) Installation Ollama:

Lade den Installer herunter:

<https://ollama.com/download>

Ollama wird über die Konsole (CLI) mit dem Befehl `ollama` aufgerufen

Verfügbare Commands:

<code>serve</code>	Start ollama
<code>create</code>	Create a model from a Modelfile
<code>show</code>	Show information for a model
<code>run</code>	Run a model
<code>stop</code>	Stop a running model

pull	Pull a model from a registry
push	Push a model to a registry
list	List models
ps	List running models
cp	Copy a model
rm	Remove a model
help	Help about any command

Flags:

-h, --help    help for ollama  
-v, --version    Show version information

### a) Modell auswählen und laden

Ollama bietet Zugriff auf eine Vielzahl an Modellen über einfache Befehle. Die Auswahl hängt von Anwendungsfall, Hardware und Zielsetzung ab. Hier eine beispielhafte Auswahl:

Modell	Befehl zum Laden	Beschreibung
neural-chat	ollama pull neural-chat	Stabil, CPU-freundlich, Intel optimiert
stablelm-zephyr	ollama pull stablelm-zephyr	Sehr gute Sprachverarbeitung
llama2:7b	ollama pull llama2:7b	Leistungsstark, benötigt mehr RAM
phi3:mini	ollama pull phi3:mini	Klein, schnell, eingeschränktes Wissen
mistral	ollama pull mistral	Kompakt, vielseitig, sehr beliebt

**Hinweis:** Die Modelle werden beim ersten Pull automatisch heruntergeladen (~1–4 GB je nach Modellgröße).

Eine volle Liste findet man unter:

<https://ollama.com/models>

### b) Modell ausführen (serve)

Sobald ein Modell geladen ist, kann es entweder direkt über die Kommandozeile genutzt oder über die REST-API angesprochen werden:

#### Beispiel CLI-Nutzung:

```
ollama run neural-chat
```

**Nutzung in Python (über langchain\_ollama):**

```

from langchain_ollama import OllamaLLM
from langchain_core.prompts import ChatPromptTemplate

# Initialisiere das LLM
model = OllamaLLM(model="neural-chat")

```

Ollama verarbeitet Prompts dabei sequentiell und liefert eine Textantwort zurück – ideal für die Integration in Skripte oder automatisierte Prüfprozesse.

### c) Modellwahl in der Praxis

Die richtige Modellwahl hängt vom jeweiligen Sicherheitsanwendungsfall ab:

Zielsetzung	Modellempfehlung
E-Mail/Phishing-Analyse	neural-chat oder mistral
Logauswertung oder CVE-Klassifikation	stablelm-zephyr oder llama2
CPU-beschränkte Umgebung, schnelle Checks	phi3:mini oder tinyllama

**Tipp:** Teste verschiedene Modelle mit identischen Prompts und vergleiche die Antworten. Dies hilft bei der Auswahl des stabilsten und verständlichsten Outputs für deine konkrete Aufgabe.

## III. Umsetzung

Dieses Kapitel beschreibt den praktischen Ablauf von der Auswahl eines konkreten Use Cases bis zur Ausführung der Analyse mit einem Small Language Model (SLM) über **Ollama**. Ziel ist die automatisierte Verarbeitung sicherheitsrelevanter Daten und deren strukturierte Bewertung.

### 1. Vorbereitung der Eingabedaten

Je nach Use Case müssen die Rohdaten zuerst in ein geeignetes Format gebracht werden. Dazu zählt:

- **Extraktion** aus Quellformaten (JSON, TXT, LOG, CSV, ...)
- **Normalisierung** (z. B. Entfernen irrelevanter Metadaten)
- **Textsegmentierung** (z. B. nur relevante Beschreibungstexte)

### 2. Modellansteuerung und Ausgabe als JSON

Im folgenden Python-Code werden die Testdaten eingelesen, durch das Modell bewertet und die Ergebnisse als neue JSON-Datei gespeichert:

```

import json
from langchain_ollama import OllamaLLM
from langchain_core.prompts import ChatPromptTemplate
from langchain_core.output_parsers import JsonOutputParser

with open("<Dateinamen>", "r", encoding="utf-8") as file:
    data = json.load(file)

```

```

# Parser für strukturierte JSON-Antworten
parser = JsonOutputParser()

Model und Prompt in für Variablen zuweisen
model =
prompt =

# Ergebnisse werden hier gesammelt
evaluated = []

# Iteriere durch die Daten
for item in data:
    formatted_prompt = prompt.invoke(item)
    response = model.invoke(formatted_prompt)
    parsed = parser.invoke(response)
    evaluated.append(parsed)

# Ausgabe in neue JSON-Datei schreiben
with open("<Zieldatei>.json", "w", encoding="utf-8") as outfile:
    json.dump(evaluated, outfile, indent=2, ensure_ascii=False)

print("✅ Auswertung abgeschlossen. Ergebnisse in '<Zieldatei>.json' gespeichert.")

```

### 3. Erstellen des richtigen Prompts

Ein gut formulierter Prompt ist entscheidend für die Qualität der Modellantworten. Besonders bei sicherheitsrelevanten Anwendungen muss der Prompt klar definieren, **was analysiert werden soll**, **wie die Ausgabeform auszusehen hat** und **welche Rolle das Modell übernimmt**.

#### 3.1 Ziel und Rolle definieren

Zu Beginn sollte der Prompt dem Modell explizit mitteilen, in welcher Rolle es agieren soll, z. B.:

- „Du bist ein Cybersicherheitsanalyst.“
- „Du bist ein Modell zur Einordnung von Schwachstellen in IT-Systemen.“
- „Du bist ein KI-gestützter Filter für sicherheitskritische E-Mails.“

Diese Rollenbeschreibung dient als *Systemkontext* und hilft dem Modell, Aufgaben richtig zu interpretieren.

#### 3.2 Aufgabenbeschreibung und Formatvorgabe

Im nächsten Schritt wird der Prompt um eine klare Aufgabenbeschreibung ergänzt, etwa:

„Bewerte die Schwere der folgenden Sicherheitslücke und gib eine strukturierte JSON-Antwort zurück. Mögliche Schweregrade: Niedrig, Mittel, Hoch.“

Ein Beispiel für eine vollständige Ausgabevorgabe:

```
{
  "schweregrad": "<Niedrig | Mittel | Hoch>",
  "begründung": "<Kurze Begründung>"
}
```

Diese explizite Formatvorgabe stellt sicher, dass die Antwort maschinenlesbar und leicht weiterverarbeitbar bleibt.

### 3.3 Beispiel-Prompt in Python

Nachfolgend ein Beispiel für einen Prompt, wie er in Python mit `langchain` umgesetzt werden kann:

```
from langchain_core.prompts import ChatPromptTemplate


template = """
Du bist ein Modell zur Einschätzung von Schwachstellen im Bereich IT-Sicherheit.

Klassifiziere die Schwere der folgenden Schwachstelle: niedrig, mittel oder hoch.
Gib eine kurze Begründung. Antworte strukturiert im JSON-Format.

Hier ist die Schwachstellenbeschreibung:
\"\"\"{input_data}\"\"\"

Antwortformat:
{{
  "schweregrad": "<Niedrig | Mittel | Hoch>",
  "begründung": "<Kurze Begründung>"
}}
"""

prompt = ChatPromptTemplate.from_template(template)
```

 **Tipp:** Verwende einfache, klare Anweisungen im Prompt und vermeide mehrdeutige Formulierungen. Nutze konkrete Beispiele, falls das Modell Schwierigkeiten beim Einhalten des Formats zeigt.

## IV. Integration

### 1. Integration in bestehende Cybersecurity-Systeme

Die Integration von SLMs in Cybersecurity-Umgebungen kann auf verschiedene Arten erfolgen:

- **Standalone-Tools:** Nutzung als eigenständige Analyseprogramme oder CLI-Werkzeuge.
- **Microservices:** Bereitstellung als Sicherheits-API, die von anderen Systemen angesprochen wird.
- **Echtzeit-Überwachung:** Automatisierte Analyse von eingehenden Nachrichten, Logs oder Netzwerkdaten in Echtzeit.

- **Benutzerunterstützung:** Einbindung in Dashboards oder SIEM-Systeme zur Unterstützung von Analysten.
- **Workflows und Automatisierung:** Integration in Incident Response oder Orchestrierungslösungen.

Die Auswahl der Integrationsmethode sollte Anforderungen wie Latenz, Skalierbarkeit und Datenschutz Rechnung tragen.

## 2. Empfehlungen und bewährte Praktiken

- **Prompt-Engineering:** Investieren Sie Zeit in die Optimierung des Prompts, um präzise und verlässliche Resultate zu erhalten.
- **Modellvergleich:** Testen Sie verschiedene SLM-Modelle hinsichtlich Genauigkeit und Performance für den spezifischen Use Case.
- **Ressourcenschonender Betrieb:** Nutzen Sie die Vorteile von SLMs zur effizienten Nutzung vorhandener Hardware.
- **Datenschutz:** Lokale Modellnutzung minimiert Risiken beim Umgang mit sensiblen Sicherheitsdaten.
- **Iterative Entwicklung:** Verfeinern Sie Prompt, Datenvorbereitung und Integration schrittweise.
- **Fehlerbehandlung:** Implementieren Sie Mechanismen zur Erkennung und Handhabung von unvorhergesehenen Modellausgaben.

## 3. Fazit

Small Language Models bieten im Cybersecurity-Bereich eine effiziente Möglichkeit, KI-gestützte Automatisierungen und Analysen mit geringem Ressourcenbedarf und hohen Datenschutzstandards zu realisieren. Durch eine systematische Herangehensweise von der Auswahl des Anwendungsfalls über die Modellwahl bis hin zur Integration lassen sich spezialisierte, wirksame Sicherheitslösungen entwickeln.