## **פרק 1: התקנת WSL + Ubuntu (לווינדוס)**

### **מה זה עושה?**

WSL מאפשר להריץ לינוקס בתוך ווינדוס – הכרחי אם יש לך מחשב ווינדוס ורוצה להריץ שרתים/AI מקומיים. נשתמש ב־Ubuntu שהיא הגרסה הכי קלה להתחלה.

### **שלב א: פתיחת PowerShell כמנהל**

1. לחץ על תפריט ה־Start.
2. כתוב **PowerShell**.
3. עמוד על **Windows PowerShell** → קליק ימני → **Run as administrator** (הפעל כמנהל).  
   * תופיע חלונית כחולה.

### **שלב ב: התקנת WSL ואובונטו**

כתוב בחלון (אפשר להעתיק/להדביק):  
  
 powershell  
CopyEdit  
wsl --install -d Ubuntu

1. לחץ **Enter**.
2. המתן (יכול לקחת כמה דקות). יופיעו הרבה שורות, זה תקין.
3. בסוף – תתבקש להפעיל את המחשב מחדש (Restart).  
    **עשה זאת!**
4. אחרי הריסטארט, בפעם הראשונה – תראה מסך של Ubuntu:  
   * המתן שוב (זה מוריד את מערכת ההפעלה).
   * בסיום – תתבקש לבחור שם משתמש (הקלד למשל: myuser, Enter).
   * תתבקש לבחור סיסמה (להקליד פעמיים).

### **שלב ג: עידכון והתקנת כלים בסיסיים**

בחלון ה־Ubuntu שייפתח כתוב:  
  
 bash  
CopyEdit  
sudo apt update && sudo apt upgrade -y

* + תתבקש להקליד סיסמה (תוך כדי הקלדה לא רואים כלום – זה תקין).
  + תן לו לסיים (יכול לקחת 2–3 דקות).

התקנת כלים בסיסיים (הכל שורה אחת!):  
  
 bash  
CopyEdit  
sudo apt install -y git curl nano htop unzip

* + שוב תן לו לרוץ עד הסוף.

## **פרק 2: התקנת Docker ו-Docker Compose על Ubuntu/WSL**

### **מה זה Docker ולמה צריך אותו?**

Docker הוא כלי שמאפשר להריץ "קונטיינרים" – סביבות עבודה מבודדות, שמריצות אפליקציות בדיוק אותו דבר בכל מחשב.  
 **בפשטות:** Docker זה כמו "קופסה" שבתוכה רצה האפליקציה שלך (למשל n8n, Open WebUI, וכו').  
 **Docker Compose** מאפשר להרים כמה קופסאות (קונטיינרים) בפקודה אחת, כולל תקשורת ביניהן.

### **מתי צריך את זה?**

* תמיד – כל עוד אתה עובד לפי המדריך הזה, גם על WSL וגם על מחשב לינוקס רגיל!

### **שלב א: ודא שהמערכת מעודכנת**

**העתק כל שורה ולחץ Enter:**

bash

CopyEdit

sudo apt update && sudo apt upgrade -y

* למה? לוודא שיש לך את הגרסאות הכי חדשות של כל החבילות.
* אמור לרוץ כמה דקות. בסוף אמור להחזיר אותך לשורת פקודה (ללא שגיאות).

### **שלב ב: התקן חבילות מקדימות**

**העתק:**

bash

CopyEdit

sudo apt install -y ca-certificates curl gnupg lsb-release

* למה צריך את זה?  
  + ca-certificates – חבילות אבטחה,
  + curl – להורדת קבצים מהאינטרנט,
  + gnupg – לאימות מפתחות,
  + lsb-release – מידע על גרסת לינוקס.

### **שלב ג: הורד מפתח חתימה של Docker**

**העתק:**

bash

CopyEdit

sudo mkdir -p /etc/apt/keyrings

curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo gpg --dearmor -o /etc/apt/keyrings/docker.gpg

* למה צריך את זה?  
   המפתח הזה יאפשר למערכת לוודא שכל החבילות שמורידים של Docker אמינות וחתומות.

### **שלב ד: הוסף את מאגר Docker למערכת**

**העתק:**

bash

CopyEdit

echo "deb [arch=$(dpkg --print-architecture) signed-by=/etc/apt/keyrings/docker.gpg] https://download.docker.com/linux/ubuntu $(lsb\_release -cs) stable" | sudo tee /etc/apt/sources.list.d/docker.list > /dev/null

* זה מוסיף את "החנות הרשמית" של Docker למערכת.

### **שלב ה: עדכן שוב את המערכת**

**העתק:**

bash

CopyEdit

sudo apt update

* עכשיו היא רואה גם את המאגר של Docker.

### **שלב ו: התקנת Docker**

**העתק:**

bash

CopyEdit

sudo apt install -y docker-ce docker-ce-cli containerd.io docker-buildx-plugin docker-compose-plugin

* **זהו – שלב ההתקנה!**
* ייקח כמה דקות (הורדה + התקנה).

### **שלב ז: בדיקת Docker**

**בדיקה 1:** הרץ פקודת בדיקה:

bash

CopyEdit

docker --version

* אמור להחזיר משהו כמו:  
   Docker version 25.0.3, build ...
* אם קיבלת שגיאה – עצור כאן ושלח לי צילום מסך.

**בדיקה 2:** הרץ את "hello world" של Docker:

bash

CopyEdit

sudo docker run hello-world

* אמור להופיע מסך מלא שמסביר שההתקנה עובדת!
* (אם מופיעה שגיאה של הרשאות, זה תקין – תתקן בשלב הבא.)

### **שלב ח: הפעלת Docker אוטומטית (רשות)**

כדי ש־Docker יופעל אוטומטית עם עליית המחשב:

bash

CopyEdit

sudo systemctl enable docker

sudo systemctl start docker

* לא חובה, אבל חוסך לך "ידנית" כל פעם.

### **שלב ט: לאפשר להריץ Docker בלי sudo (אופציונלי, נוח מאוד!)**

הוסף את המשתמש שלך לקבוצת docker:  
  
 bash  
CopyEdit  
sudo usermod -aG docker $USER

**חשוב:** תתנתק מהטרמינל ותתחבר שוב (או תבצע exit ותחזור).  
 או תכתוב:  
  
 bash  
CopyEdit  
newgrp docker

* עכשיו אפשר להריץ docker בלי sudo.

### **שלב י: התקנת Portainer (לא חובה, מומלץ למתחילים)**

Portainer זה ממשק גרפי (בדפדפן) לניהול Docker.

צור volume:  
  
 bash  
CopyEdit  
docker volume create portainer\_data

הרץ Portainer:  
  
 bash  
CopyEdit  
docker run -d -p 9000:9000 --name=portainer --restart=always \

-v /var/run/docker.sock:/var/run/docker.sock \

-v portainer\_data:/data portainer/portainer-ce

1. גש לדפדפן בכתובת:  
   <http://localhost:9000>
2. תגדיר משתמש/סיסמה – ותוכל לנהל קונטיינרים בקלות.

### **דגשים ותקלות נפוצות**

* **שגיאת הרשאה:** אם אתה מקבל permission denied כשאתה מריץ docker, ודא שסיימת את שלב ט' (הוספת משתמש ל־docker) והתנתקת מהטרמינל וחזרת.
* **בעיה עם systemd ב־WSL?** בד"כ Docker עובד גם בלי systemd, אבל אם משהו לא עובד – כתוב לי!

**סיימת פרק 2 – Docker מוכן!** עבור לפרק הבא: **התקנת Ollama, הורדת מודלים ובדיקות**.

### **שלב א: מחיקת Ollama ישן (אם התקנת בעבר)**

*לא חובה – רק אם התקנת ניסית בעבר או יש לך שאריות קודמות.*

bash

CopyEdit

sudo systemctl stop ollama

sudo rm -rf /usr/local/bin/ollama /usr/local/lib/ollama ~/.ollama /etc/systemd/system/ollama.service

sudo systemctl daemon-reload

* **הערה:** שורה זו מוחקת את Ollama וכל ההגדרות הישנות. אל תדאג – זה לא מוחק כלום חשוב אחר.

### **שלב ב: התקנת Ollama חדש**

1. **העתק את הפקודה והרץ בטרמינל:**

bash

CopyEdit

curl -fsSL https://ollama.com/install.sh | sh

1. **כאשר תתבקש – הזן את סיסמת המשתמש שלך (sudo).**
2. בסיום התקנה תקבל הודעה:  
    "The Ollama API is now available at 127.0.0.1:11434"

### **שלב ג: בדיקת גרסה**

בדוק שההתקנה הצליחה:

bash

CopyEdit

ollama --version

* אמור להופיע משהו כמו:  
   ollama version 0.1.37

### **שלב ד: בדיקת שירות Ollama**

בדוק אם Ollama רץ:

bash

CopyEdit

sudo systemctl status ollama

* אמור להופיע שהוא **active (running)**.

לא רץ? הרץ:  
  
 bash  
CopyEdit  
sudo systemctl restart ollama

### **שלב ה: בדיקת זמינות API (פורט 11434)**

בדוק שה-API זמין:

bash

CopyEdit

curl http://localhost:11434

* אמור להחזיר:  
   Ollama is running

### **שלב ו: הורדת מודל שיחה ראשון**

**Aya – מודל שיחה בעברית**

bash

CopyEdit

ollama pull aya

* הורדה ראשונית – תלוי באינטרנט, אמור להוריד כמה מאות מגה-בייט.
* לאחר סיום – תוכל לראות את המודל ברשימת המודלים.

*מודלים נוספים (אם תרצה):*

bash

CopyEdit

ollama pull mistral

ollama pull llama3

* לא חובה, רק אם תרצה מודלים באנגלית/מודלים מתקדמים.

### **שלב ז: בדיקת רשימת מודלים**

bash

CopyEdit

ollama list

* תראה את כל המודלים שהורדת – אמור להופיע aya, ועוד.

### **שלב ח: הרצת מודל ראשון – בדיקת שיחה**

נסה שיחה ישירה מהטרמינל:

bash

CopyEdit

ollama run aya

* תוכל לכתוב שאלה – למשל:  
   מה שלומך?
* אמור להחזיר תשובה בעברית.

### **שלב ט: בדיקת API עם Prompt (בודקים שהכל פועל דרך API)**

bash

CopyEdit

curl http://localhost:11434/api/generate -d '{"model": "aya", "prompt": "שלום, מה שלומך?"}'

* אמור להחזיר תשובה בפורמט JSON, עם טקסט של המודל.

### **שלב י: הורדת מודל אימבדינג (חובה ל-RAG)**

מודל אימבדינג (ליצירת וקטורים למסמכים):

bash

CopyEdit

ollama pull mxbai-embed-large

* מודל יחסית קטן (עשרות עד מאות מגה).
* חובה להריץ אותו לפני שימוש ב־RAG.

### **שלב יא: בדיקת הרשאה (האם Ollama רץ כ־Daemon ברקע)**

בדוק שיש תהליך בשם ollama:

bash

CopyEdit

ps aux | grep ollama

* אמור להופיע שורה עם /usr/local/bin/ollama serve

אם לא – נסה להריץ שוב:  
  
 bash  
CopyEdit  
sudo systemctl restart ollama

### **שלב יב: הערות ודגשים**

אם קיבלת שגיאת **binding address** –  
 כנראה יש לך שירות אחר על פורט 11434, או ש-Ollama כבר רץ.  
 אפשר לעצור ולהפעיל מחדש:  
  
 bash  
CopyEdit  
sudo systemctl stop ollama

sudo systemctl start ollama

* **אם אתה מריץ על WSL:** ודא שאין אנטי-וירוס/חומת אש שחוסמת את הפורט.

**סיימת פרק 3 – Ollama מוכן, המודלים זמינים, הכל נבדק!**

## **פרק 3.א: הפעלת Ollama על כל הכתובות באופן קבוע**

### **למה צריך את זה?**

* כברירת מחדל, Ollama מאזין רק ל־localhost (127.0.0.1) –  
   כלומר, רק תוכנות שרצות על אותו מחשב יוכלו לדבר איתו.
* אם אתה מריץ כלים בתוך Docker (n8n, WebUI), או ממחשב אחר ברשת –  
   חובה ש־Ollama יאזין ל־0.0.0.0 (כל הכתובות/כל הרשת).
* הפקודה הזמנית (OLLAMA\_HOST=0.0.0.0 ollama serve) עובדת רק כל עוד הטרמינל פתוח –  
   רוצים שזה יהיה קבוע? משנים את הקובץ של systemd.

### **שלב א: עריכת שירות Ollama**

פתח את קובץ השירות של Ollama לעריכה:  
  
 bash  
CopyEdit  
sudo nano /etc/systemd/system/ollama.service

חפש את השורה שמתחילה ב־**ExecStart=** לדוגמה:  
  
 ini  
CopyEdit  
ExecStart=/usr/local/bin/ollama serve

שנה אותה כך:  
  
 swift  
CopyEdit  
ExecStart=/usr/bin/env OLLAMA\_HOST=0.0.0.0 /usr/local/bin/ollama serve

1. **שים לב:** אל תשכח /usr/bin/env לפני ה־OLLAMA\_HOST!
2. שמור את הקובץ:  
   * לחץ Ctrl+O, ואז Enter
   * ואז Ctrl+X ליציאה

### **שלב ב: טעינה מחדש של systemd**

bash

CopyEdit

sudo systemctl daemon-reload

### **שלב ג: הפעלת השירות מחדש**

bash

CopyEdit

sudo systemctl restart ollama

### **שלב ד: בדיקת שהכל תקין**

בדוק שהשירות מאזין על כל הכתובות (לא רק על 127.0.0.1):

bash

CopyEdit

ss -tuln | grep 11434

אמור להופיע:  
  
 nginx  
CopyEdit  
tcp LISTEN 0 4096 0.0.0.0:11434 0.0.0.0:\*

או:  
  
 markdown  
CopyEdit  
tcp LISTEN 0 4096 \*:11434 \*:\*

* אפשר לבדוק גם ב־hostname -I (לקבל כתובת IP שלך).

### **שלב ה: הערות ודגשים**

* **זהו שלב חשוב!** אם דילגת עליו – Docker וכלים חיצוניים לא יצליחו לדבר עם Ollama.
* אם תרצה לחזור למצב המקורי (localhost בלבד) –  
   ערוך שוב את הקובץ והסר את חלק ה־OLLAMA\_HOST.

### **דוגמה: מה קורה אם לא עשית את זה?**

* תנסה מה־Docker להתחבר ל־Ollama – תקבל שגיאת חיבור (connection refused).
* תנסה מה־n8n – תקבל שגיאה דומה.

**סיימת פרק 3.א! Ollama מוכן לכלים חיצוניים, Docker ו־WSL.**

## **פרק 4: התקנת Open WebUI – צ’אט עם Ollama בדפדפן**

### **מה זה Open WebUI?**

Open WebUI הוא ממשק גרפי מודרני שמאפשר לנהל ולדבר עם Ollama דרך הדפדפן:

* לנהל ולהריץ מודלים (למשוך, למחוק, להגדיר ברירת מחדל)
* לכתוב פקודות ושאלות (צ’אט)
* לשמור היסטוריית שיחות
* להגדיר System Prompts בקלות
* לתמוך בשפות שונות, כולל עברית

### **למה צריך את זה?**

* אם אתה רוצה לבדוק שהמודלים שלך עובדים, לראות תוצאות ולשלוט ב-Ollama בנוחות, Open WebUI עושה את הכל בקליק.

### **שלב 1: ודא ש־Docker מותקן ורץ**

אם עדיין לא מותקן, התקן לפי פרק 2.

### **שלב 2: ודא ש־Ollama רץ, פתוח ל־0.0.0.0 (כל הכתובות)**

בדוק שפקודת:

bash

CopyEdit

ss -tuln | grep 11434

מציגה ש-Ollama מאזין על כל הכתובות, לא רק 127.0.0.1.

### **שלב 3: הרץ את Open WebUI בדוקר**

**הכי קל להריץ עם הרשת של המחשב (network=host):**

bash

CopyEdit

docker run -d \

--name open-webui \

--restart always \

--network=host \

-v open-webui:/app/backend/data \

-e OLLAMA\_BASE\_URL=http://127.0.0.1:11434 \

ghcr.io/open-webui/open-webui:main

אם Ollama רץ על IP אחר (לדוג׳ 192.168.1.33) – עדכן ב־OLLAMA\_BASE\_URL:  
  
 bash  
CopyEdit  
-e OLLAMA\_BASE\_URL=http://192.168.1.33:11434

* ב־WSL: *חובה* להשתמש ב־--network=host כדי לאפשר תקשורת ישירה!

### **שלב 4: בדוק שהקונטיינר רץ**

bash

CopyEdit

docker ps

* אתה אמור לראות את open-webui במצב Up.

### **שלב 5: פתח את Open WebUI בדפדפן**

* עבור אל:<http://localhost:3000>
* במחשב אחר ברשת:  
   http://<IP-של-המחשב-שלך>:3000

### **שלב 6: הגדר משתמש וסיסמה (אם מתבקש)**

* יש לבחור משתמש וסיסמה בפעם הראשונה.  
   (אפשר לנהל משתמשים, לשנות פרופיל, להוסיף System Prompt כברירת מחדל וכו’.)

### **פתרון תקלות נפוצות:**

* **מודלים לא מוצגים או לא ניתן להתחבר?** ודא שה־OLLAMA\_BASE\_URL נכון ומתאים לכתובת של Ollama.

**פורט 3000 תפוס?** שנה להרצה על פורט אחר:  
  
 bash  
CopyEdit  
docker run -d ... -p 3010:3000 ...

* וגש ל־<http://localhost:3010>.

**בדוק לוגים:** bash  
CopyEdit  
docker logs open-webui

* **בעיות הרשאות / הרשמה:** נסה למחוק את הקונטיינר (docker rm -f open-webui) ולהריץ מחדש.

### **שליטה בקונטיינר (הפעלה, הפסקה, מחיקה):**

bash

CopyEdit

docker stop open-webui

docker start open-webui

docker rm open-webui

### **דגשים:**

* הממשק תומך בצ’אט רב-שפות, כולל עברית.
* הממשק *לא* מחייב חשבון – הריצה היא לוקאלית בלבד.
* אתה יכול להעלות, למחוק, ולהפעיל מודלים של Ollama ישירות מהממשק.

**סיימת את פרק 4! יש לך Ollama עם ממשק גרפי – אפשר להתחיל לבדוק מודלים ולבנות**

## **פרק 5: פתיחת בוט טלגרם חדש וחיבור ל־n8n**

### **שלב 1: יצירת בוט טלגרם חדש**

1. פתח את אפליקציית **Telegram** וחפש למעלה:  
    **BotFather**

שלח את הפקודה:  
  
 bash  
CopyEdit  
/start

ואז:  
  
 bash  
CopyEdit  
/newbot

1. בחר שם לבוט שלך (למשל: MySmartAgent).
2. בחר username שיסתיים ב־**bot** (למשל: smart\_agent\_2025bot).
3. קבל מה־BotFather **API Token** –  
    ***שמור אותו במקום בטוח*** (תצטרך להזין אותו ב־n8n).

### **שלב 2: הפעל את הבוט החדש**

* שלח הודעה לבוט (כדי להפעיל אותו).
* תוכל גם להוסיף אותו לקבוצות, אם תרצה.

### **שלב 3: כניסה ל־n8n (ממשק דפדפן)**

* פתח את הדפדפן ועבור לכתובת:  
  <http://localhost:5678> *(אם אתה מריץ n8n על מחשב אחר – כתובת ה־IP שלו)*

### **שלב 4: יצירת Credentials (הרשאות) לבוט טלגרם**

1. עבור ל־**Credentials** (למעלה בתפריט או תחת Settings).
2. הוסף סוג חדש: **Telegram API**.
3. הדבק את ה־API Token שקיבלת מה־BotFather.
4. שמור.

### **שלב 5: בניית Workflow בסיסי ב־n8n לקבלת הודעות מהבוט**

1. **צור Workflow חדש** – לחץ על "New Workflow".
2. **הוסף Node ראשון:** חפש "Telegram" ובחר **Telegram Trigger**.
3. בחר את ה־Credentials שהגדרת.
4. בחר Event:  
   * הכי נפוץ: **Message** (מגיב לכל הודעה שמתקבלת).
5. לחץ על **Execute Node** כדי לבדוק –  
    שלח הודעה לבוט ובדוק שה-Node קולט אותה.

### **שלב 6: שליחת תשובה חזרה בטלגרם**

1. **הוסף Node:** "Telegram > Send Message"

ב־**Chat ID**:  
  
 scss  
CopyEdit  
{{ $json["message"]["chat"]["id"] }}

ב־**Text**:  
 כתוב משהו פשוט (לבדיקה), למשל:  
  
 CopyEdit  
קיבלתי את ההודעה שלך!

1. חבר בין ה־Nodes (מה־Trigger ל־Send Message).
2. לחץ שוב על **Execute Workflow** ושלח הודעה לבוט.
3. בדוק שהבוט עונה אוטומטית בטלגרם!

### **טיפים חשובים:**

* שים לב לשמות ה־nodes: אם שינית שמות או מבנה, הנתיב לשדה chat.id עשוי להשתנות.
* אל תשתף את ה־API Token עם אף אחד.
* אפשר להגדיר ב־n8n בוטים נוספים, להפעיל כמה Workflows במקביל וכו’.

**ברכות! סיימת את פרק 5 – יש לך בוט טלגרם מחובר ל־n8n שמקבל ומחזיר הודעות אוטומטיות!**

## **פרק 6: בניית Workflow חכם ב־n8n – סוויץ' קול/טקסט, תמלול ושיחה עם AI**

### **מה נבנה בפרק הזה?**

* **Workflow מתקדם** שמסוגל:  
  1. לזהות האם המשתמש שלח הודעת טקסט או הודעה קולית.
  2. אם זו הודעה קולית – לשמור את הקובץ, לתמלל אותו (whisper), ולהפוך אותו לטקסט.
  3. אם זו הודעת טקסט – להעביר את התוכן ישירות.
  4. להעביר את התוכן (קול או טקסט) למודל ה־AI (Ollama Agent).
  5. להחזיר תשובה אוטומטית למשתמש בטלגרם.

### **שלב 1: יצירת Workflow חדש**

1. פתח את n8n ולחץ על **New Workflow**.

### **שלב 2: Node ראשון – Telegram Trigger**

* כמו בפרק הקודם (Message Event).

### **שלב 3: Node שני – Switch**

לבדוק אם זו הודעה קולית או טקסט

1. חפש והוסף Node בשם **Switch**.

בחר **Value 1**:  
  
 scss  
CopyEdit  
{{$json["message"]["voice"]["file\_id"]}}

1. בחר **Operation**:  
   * **Is not empty** (אם יש קובץ קול)
2. הגדר שני מסלולים:  
   * אם יש file\_id => מסלול קול (Voice)
   * אחרת => מסלול טקסט (Text)

### **שלב 4: מסלול קול (Voice)**

#### **4.1 Telegram Get File**

* משמש להוריד את קובץ הקול.
* הגדר file\_id מה-Trigger.

#### **4.2 Write Binary File**

* שומר את הקובץ בתיקייה קבועה ב־volume של Docker (למשל /audio/voice.ogg).

#### **4.3 Execute Command**

* מריץ את whisper (סקריפט transcribe.sh) כדי לתמלל את הקובץ לקובץ טקסט (voice.txt).

#### **4.4 Edit Fields**

* קורא את הטקסט המפוענח מהפלט (stdout או מתוך קובץ הטקסט).
* מכניס את התמלול לשדה שנקרא "text" (להמשך).

### **שלב 5: מסלול טקסט (Text)**

**Edit Fields** – יוצר שדה בשם "text" מתוך  
  
 scss  
CopyEdit  
{{$json["message"]["text"]}}

### **שלב 6: איחוד מסלולים**

**(לא חובה, אפשר פשוט להעביר שניהם ל־Agent):**

* כל מסלול (קול או טקסט) יוצא עם שדה "text".
* כל Node אחר (Agent) ייקח תמיד מהשדה הזה.

### **שלב 7: AGENT – Node AI (Ollama)**

* **הוסף Node חדש:**
  + תלוי מה מותקן אצלך:  
    - Ollama Agent
    - HTTP Request ל־FastAPI

בשדה ה־Prompt:  
  
 bash  
CopyEdit  
ענה תמיד בעברית: {{$json["text"]}}

* + אפשר להגדיר System Prompt נוסף אם יש.

### **שלב 8: שליחת תשובה חזרה בטלגרם**

* **Telegram > Send Message**

**Chat ID:** scss  
CopyEdit  
{{$json["message"]["chat"]["id"]}}

**Text:** bash  
CopyEdit  
{{$json["result"] || $json["answer"]}}

* + שים לב לשם השדה! (אם קוראים לו אחרת – לעדכן בהתאם).

### **שלב 9: בדיקה**

1. **שמור את ה-Workflow והפעל אותו**.
2. שלח הודעת טקסט – קבל תשובה חכמה.
3. שלח הודעה קולית – תקבל תשובה (אחרי שהתמלול הסתיים).

### **טיפים ל־n8n:**

* תוכל תמיד לבדוק כל Node ע"י "Execute Node" ו"לראות Output" (מומלץ במיוחד כשהנתיב לשדה משתנה).
* אפשר להרחיב למסלולים נוספים (תמונה, מסמך, REST וכו').
* שמור על שמות קבצים ותיקיות קבועות – במיוחד כשעובדים עם Volumes.
* אם משהו לא עובד – בדוק לוגים, תן שמות ברורים ל־Nodes ובדוק שמות שדות.

**זהו! יש לך Workflow חכם שמנהל תקשורת עם הבוט, כולל תמלול ודיבור עם AI.**

## **פרק 7: התקנת whisper.cpp לתמלול קולי (ב־Docker בלבד!)**

### **למה צריך את זה?**

* **whisper.cpp** הוא כלי תמלול קולי (Speech-to-Text) שרץ לגמרי מקומית, בחינם וללא שימוש בענן.
* חובה להריץ אותו *בתוך* קונטיינר הדוקר של n8n — אחרת לא תוכל לגשת אליו ב-Workflow שלך!
* מתאים לכל קובץ קול שמגיע מהמשתמש (בעיקר קבצי OGG/MP3 מטלגרם).

### **שלב 1: הכנת סביבת העבודה**

*(הכול מתוך WSL/Ubuntu)*

bash

CopyEdit

mkdir ~/n8n-docker

cd ~/n8n-docker

### **שלב 2: שכפול והידור whisper.cpp**

bash

CopyEdit

git clone https://github.com/ggerganov/whisper.cpp.git

cd whisper.cpp

make

### **שלב 3: הורדת מודל תמלול (model)**

bash

CopyEdit

cd ~/n8n-docker/whisper.cpp

./models/download-ggml-model.sh base

תוכל להחליף ל־base.en לאנגלית בלבד, או דגם גדול יותר לשפות נוספות.

### **שלב 4: הכנת תיקיות משותפות**

bash

CopyEdit

cd ~/n8n-docker

mkdir -p n8n\_data/whisper/build/bin

mkdir -p n8n\_data/whisper/models

mkdir voice-temp

### **שלב 5: העתקת קבצים**

bash

CopyEdit

cp whisper.cpp/build/bin/whisper-cli n8n\_data/whisper/build/bin/

cp whisper.cpp/models/ggml-base.bin n8n\_data/whisper/models/

ודא ששמות הקבצים תואמים — אם הורדת דגם אחר (למשל base.en), העתק בהתאמה!

### **שלב 6: יצירת סקריפט transcribe.sh**

bash

CopyEdit

nano n8n\_data/transcribe.sh

הדבק את התוכן הבא (התאם נתיבים למודל/בינארי במידת הצורך):

bash

CopyEdit

#!/bin/sh

AUDIO\_FILE="$1"

WAV\_FILE="/audio/voice.wav"

TXT\_FILE="/audio/voice.txt"

WHISPER\_BIN="/home/node/.n8n/whisper/build/bin/whisper-cli"

MODEL\_PATH="/home/node/.n8n/whisper/models/ggml-base.bin"

ffmpeg -y -i "$AUDIO\_FILE" -ar 16000 -ac 1 -f wav "$WAV\_FILE"

$WHISPER\_BIN -m "$MODEL\_PATH" -f "$WAV\_FILE" -otxt -of "/audio/voice"

cp /audio/voice.wav.txt "$TXT\_FILE"

echo "Transcription complete: $TXT\_FILE"

שמור (Ctrl+O, Enter, Ctrl+X).

### **שלב 7: הרשאות הפעלה**

bash

CopyEdit

chmod +x n8n\_data/transcribe.sh

chmod +x n8n\_data/whisper/build/bin/whisper-cli

### **שלב 8: הגדרת Volumes ב־docker-compose**

ב־docker-compose.yml ודא שיש:

yaml

CopyEdit

volumes:

- ./voice-temp:/audio

- ./n8n\_data:/home/node/.n8n

(בתוך ה־service של n8n)

### **שלב 9: הרצת הקונטיינר**

bash

CopyEdit

docker compose up -d

### **שלב 10: התקנות ובדיקות בתוך הקונטיינר**

**כניסה לקונטיינר כ־root:** bash  
CopyEdit  
docker exec -u 0 -it n8n /bin/sh

**התקן כלים בסיסיים:** bash  
CopyEdit  
apk add --no-cache build-base git ffmpeg nano

**ודא הרשאות על הקבצים:** bash  
CopyEdit  
chmod +x /home/node/.n8n/transcribe.sh

chmod +x /home/node/.n8n/whisper/build/bin/whisper-cli

### **שלב 11: בדיקת תמלול**

bash

CopyEdit

sh /home/node/.n8n/transcribe.sh /audio/voice.ogg

cat /audio/voice.txt

התוצאה: הטקסט המתומלל מתוך הודעת הקול.

### **מקורות:**

* whisper.cpp:<https://github.com/ggerganov/whisper.cpp>
* מדריך תמלול לדוקר:<https://www.youtube.com/watch?v=B1eTn2GfCFc>
* תיעוד n8n:<https://docs.n8n.io/>
* תיעוד Volumes:<https://docs.docker.com/storage/volumes/>

**סיימת עם התקנת whisper! עכשיו כל הודעת קול בטלגרם תוכל להיות מתומללת אוטומטית ולשלב אותה ב־Workflow שלך.**

## **פרק 8: התקנת ngrok (NGROK) לקבלת Webhook חיצוני ל־n8n**

### **למה צריך את זה?**

* **ngrok** הוא כלי שמאפשר לך לפתוח "מנהרה" (Tunnel) זמנית מהאינטרנט למחשב שלך.
* זה פותר את בעיית הנגישות של n8n כשהוא רץ מקומית (מאחורי NAT, נתב, Firewall וכו').
* כל שירות חיצוני (כמו Telegram, WhatsApp, Google וכו') יוכל להפעיל אצלך Webhook – אפילו אם אין לך IP קבוע!

### **שלב 1: הרשמה ל־ngrok (בחינם)**

1. פתח דפדפן:  
   <https://ngrok.com/>
2. צור חשבון.
3. קבל את ה־Authtoken (תמצא אותו בלוח הניהול תחת Setup & Installation).

### **שלב 2: התקנת ngrok**

*ב־Ubuntu/WSL*

bash

CopyEdit

curl -s https://ngrok-agent.s3.amazonaws.com/ngrok.asc | sudo tee /etc/apt/trusted.gpg.d/ngrok.asc >/dev/null

echo "deb https://ngrok-agent.s3.amazonaws.com buster main" | sudo tee /etc/apt/sources.list.d/ngrok.list

sudo apt update && sudo apt install ngrok

### **שלב 3: קונפיגורציה של ה־Authtoken**

bash

CopyEdit

ngrok config add-authtoken <ה־authtoken שלך>

לדוג' (העתק בדיוק כפי שקיבלת מהאתר)

### **שלב 4: הרצת Tunnel ל־n8n**

bash

CopyEdit

ngrok http 5678

* תראה כתובת כמו:  
   **https://xxxx-xxx-xx.ngrok-free.app**

### **שלב 5: עדכון WEBHOOK\_URL בקובץ docker-compose.yml**

1. פתח את הקובץ (docker-compose.yml) לעריכה:
2. הוסף או עדכן את השורה (בתוך environment של n8n):

yaml

CopyEdit

environment:

- WEBHOOK\_URL=https://xxxx-xxx-xx.ngrok-free.app/

החלף לכתובת שקיבלת מ-ngrok!

1. שמור וסגור.

### **שלב 6: הפעל מחדש את n8n כדי שה־WEBHOOK\_URL יתעדכן**

bash

CopyEdit

docker compose down

docker compose up -d

### **שלב 7: בדיקה שה־Webhook נגיש מבחוץ**

* נסה להיכנס לכתובת מהדפדפן (אפשר גם מהטלפון/מחשב אחר).
* או שלח הודעה לבוט טלגרם שלך – אם הכל עובד, ההודעה תגיע ל־n8n שלך גם מבחוץ!

### **הערות חשובות**

* **ngrok חינמי:** הכתובת מתחלפת בכל הפעלה מחדש.  
   צריך לעדכן מחדש ב־docker-compose.
* **ngrok בתשלום:** אפשר לקבל כתובת קבועה (מומלץ לשימוש מסחרי/ארוך טווח).
* אפשר להריץ ngrok גם ישירות בקונטיינר (מתקדם, לא חובה).
* מדריך ngrok:<https://ngrok.com/docs>

### **דגשים**

* בכל פעם ש־ngrok נותן כתובת חדשה — חובה לעדכן ולהפעיל מחדש את n8n!
* אפשר לקבל עד 4 חיבורים בו־זמנית בגרסה החינמית.

**מעכשיו, כל שירות חיצוני (כמו טלגרם) יוכל לגשת ל־n8n שלך גם בלי שתפתח פורטים או תגדיר NAT!**

### **עדכון חשוב: קישור n8n לחשיפה חיצונית עם NGROK**

### **🔁 שלב ביניים: התקנת ngrok וחיבור WEBHOOK ל־n8n**

#### **1. התקן והרץ NGROK:**

אם עדיין לא עשית:

bash

CopyEdit

sudo apt update && sudo apt install ngrok

ngrok config add-authtoken <ה־authtoken שלך>

ngrok http 5678

#### **2. קבל את כתובת ה־ngrok:**

הכתובת שמתקבלת אחרי ngrok http 5678 תיראה כך:

arduino

CopyEdit

https://xxxxxx.ngrok-free.app

#### **3. עדכן את קובץ ה־docker-compose.yaml של n8n:**

ב־section של environment, שנה/הוסף שורה:

yaml

CopyEdit

environment:

- WEBHOOK\_URL=https://xxxxxx.ngrok-free.app

#### **4. בצע restart לקונטיינר:**

bash

CopyEdit

docker compose down

docker compose up -d

(או: docker restart n8n אם מריץ ישירות)

#### **5. כניסה ל־n8n**

גש לכתובת ה־ngrok שקיבלת, לדוג׳:

arduino

CopyEdit

https://xxxxxx.ngrok-free.app

(לא localhost!)

#### **דגשים:**

* **כל פעם ש־ngrok יוצר כתובת חדשה (כל ריסט/כיבוי), יש לעדכן את קובץ ה־YAML ולהריץ מחדש את הקונטיינר!**
* ב־n8n, כל Webhook שנוצר – הכתובת החיצונית שממנה מאזין תהיה זו של ngrok, לא localhost.

### **סיכום עדכון:**

* **ה־Workflow של n8n יהיה זמין רק בכתובת ngrok שהוקצתה.**
* עדכון נכון של קובץ YAML הוא קריטי לפעולה תקינה.
* העבודה תמיד מהקישור החיצוני – לא מ־localhost.

### **שלב 4: עדכן את קובץ docker-compose.yaml**

**דוגמה מלאה (בהדגשה על ה־WEBHOOK\_URL):**

yaml

CopyEdit

version: "3.8"

services:

n8n:

image: n8nio/n8n

container\_name: n8n

restart: always

ports:

- "5678:5678"

environment:

- N8N\_HOST=localhost

- N8N\_PORT=5678

- N8N\_PROTOCOL=http

- WEBHOOK\_URL=https://1a2b-34-56-78-90.ngrok-free.app

volumes:

- ./voice-temp:/audio

- ./n8n\_data:/home/node/.n8n

🟠 **שנה את הכתובת בשורה של WEBHOOK\_URL לכתובת שקיבלת מ־NGROK!** 🟠 **אם מריץ שירותים נוספים (כמו whisper וכו'), ה־volumes יהיו דומים.**

### **שלב 5: שמור וסגור את הקובץ**

(ב־nano: Ctrl+O, Enter, Ctrl+X)

### **שלב 6: הרץ מחדש את docker compose**

bash

CopyEdit

docker compose down

docker compose up -d

## **פרק 9: בניית Workflow חכם ב־n8n (סוכן קול+טקסט מבוסס Ollama)**

### **מה נעשה כאן?**

* נבנה אוטומציה מלאה שמקבלת הודעה בוט טלגרם (קול/טקסט), מתמללת (אם קול), ושולחת לשיחה במודל Ollama מקומי – ואז מחזירה תשובה אוטומטית למשתמש בטלגרם.

### **שלב 1: פתיחת Workflow חדש**

1. פתח את n8n בדפדפן:  
     
    https://כתובת-ה-ngrok-החדשה.ngrok-free.app
2. לחץ על **"New Workflow"**.

### **שלב 2: הוספת Trigger — Telegram Trigger**

* חפש "Telegram" והוסף node מסוג **Telegram Trigger**.
* צור חיבור חדש (Credentials) עם הטוקן של הבוט שיצרת.
* בחר Event: **Message**.

### **שלב 3: יצירת Switch — זיהוי קול / טקסט**

* הוסף node מסוג **Switch**.
* בחר תנאי:  
   אם קיים $json["message"]["voice"]["file\_id"] → זה קול.  
   אחרת — זה טקסט.

### **שלב 4: מסלול קול (Voice)**

1. **Telegram Get File**
   * מקבל את קובץ הקול.
2. **Write Binary File**
   * שומר את קובץ הקול ל־/audio/voice.ogg (על ה-volume של הדוקר).
3. **Execute Command**

מריץ את הסקריפט:  
  
 sh  
CopyEdit  
sh /home/node/.n8n/transcribe.sh /audio/voice.ogg

1. **Edit Fields**
   * קרא את קובץ התמלול (/audio/voice.txt) לשדה חדש בשם text או stdout.

### **שלב 5: מסלול טקסט (Text)**

* **Edit Fields**
  + שים את $json["message"]["text"] בשדה חדש בשם text.

### **שלב 6: איחוד — Merge**

* חבר את שני המסלולים (קול/טקסט) ל־node משותף (אפשרי גם בלי node Merge, תלוי איך בנוי ה־Workflow שלך).
* ודא ששדה הטקסט הסופי (תמלול או הודעה) תמיד נקרא text.

### **שלב 7: AGENT — Chat Model (Ollama)**

* הוסף node מסוג **AI Agent** או **Chat Model**.
* קבע את ה־Base URL של Ollama לכתובת הנכונה (IP שלך:11434).  
   לדוג':<http://172.18.123.44:11434>
* בחר מודל: aya

בפרומפט כתוב:  
  
 bash  
CopyEdit  
ענה תמיד בעברית. השאלה: {{$json["text"] || $json["stdout"]}}

### **שלב 8: שליחת תשובה בטלגרם**

* הוסף node מסוג **Telegram Send Message**.
* ב־Chat ID:  
   {{ $("Telegram Trigger").item.json.message.chat.id }}
* ב־Text:  
   {{ $json.output || $json.answer }}
* שמור וחבר נכון.

### **שלב 9: בדיקות**

* לחץ **Execute Workflow**.
* שלח הודעת קול/טקסט לבוט.
* קבל תשובה חכמה — קול יתומלל, טקסט יטופל מיד!

### **דגשים**

* כל הנתיבים (Volumes) של אודיו ותמלול חייבים להיות משותפים בין הדוקר ל־host.
* מומלץ לבדוק ב־"View Output" את שמות השדות בכל node.
* שים לב — כל שמות השדות יכולים להשתנות לפי המבנה, תעדכן בהתאם!

### **הרחבות אפשריות**

* הוספת תמיכה בתמונות, קבצים, REST API, וואטסאפ, מייל, Google Calendar, OCR ועוד.
* שמירה של לוגים.
* הגבלת גודל/כמות הודעות.
* שילוב RAG או מודל מתקדם לניתוב (ראו פרקים הבאים).

**סיימנו Workflow חכם שמבין קול וטקסט, מתמלל, עונה בעברית – ומשיב אוטומטית בטלגרם!**

חידוד מסלולים:

### **שלב 1: פתח את n8n בממשק הדפדפן**

(השתמש בכתובת NGROK שקיבלת, למשל: https://xxxx.ngrok-free.app)

### **שלב 2: צור Workflow חדש**

* לחץ על **"New Workflow"**
* תן שם ברור (למשל: "Smart Telegram AI Bot")

### **שלב 3: הוסף Trigger של Telegram**

1. לחץ על "+" → בחר **Telegram** → **Trigger**
2. צור **Credentials** עם ה-API Token של הבוט שלך (אם עדיין לא עשית)
3. בחר **Event:** Message
4. שמור ו**הפעל (Execute Node)**
5. שלח הודעה לבדוק שהטריגר קולט (תראה אותה ב־n8n)

### **שלב 4: הוסף Switch**

* "+" → **Switch**
* תן שם: "Voice or Text Switch"
* הגדר:  
  + תנאי:  
    - אם קיים (Exists) — message.voice.file\_id  
       → זה אומר שזה הודעת קול, אחרת זה טקסט
* חבר את ה־Switch ל־Telegram Trigger

### **שלב 5: מסלול קול (Voice Path)**

#### **5.1 Telegram Get File**

* "+" → Telegram → **Get File**
* קלט: file\_id מהטריגר (message.voice.file\_id)

#### **5.2 Write Binary File**

* "+" → **Write Binary File**
* הגדר קובץ יעד: /audio/voice.ogg

#### **5.3 Execute Command**

* "+" → **Execute Command**

פקודה:  
  
 bash  
CopyEdit  
sh /home/node/.n8n/transcribe.sh /audio/voice.ogg

#### **5.4 Edit Fields (להוציא טקסט מהפלט של whisper)**

* "+" → **Edit Fields**
* צור שדה בשם text
* מקור: שדה הפלט (stdout או voice.txt)

### **שלב 6: מסלול טקסט (Text Path)**

#### **6.1 Edit Fields**

* "+" → **Edit Fields**
* צור שדה בשם text
* מקור: message.text מהטריגר

### **שלב 7: איחוד המסלולים**

* אם צריך, השתמש ב־Merge/IF/NoOp כדי לאחד את שני המסלולים אל Node משותף (Agent).

### **שלב 8: Agent (Ollama)**

* "+" → **Ollama Chat Model** או **AI Agent** (או HTTP אם מחובר ל־RAG/שרת חיצוני)

Prompt:  
  
 bash  
CopyEdit  
תמיד ענה בעברית. שים לב: {{$json["text"]}}

* כתובת שרת (Base URL): כתובת ה־Ollama שלך (למשל: http://localhost:11434 או לפי ההתקנה)

### **שלב 9: Telegram Send Message**

* "+" → **Telegram** → **Send Message**

**Chat ID:** scss  
CopyEdit  
{{$json["message"]["chat"]["id"]}}

* (או בהתאם לנתיב בשדות שלך)

**Text:** bash  
CopyEdit  
{{$json["result"] || $json["answer"]}}

* (תעדכן לשדה הנכון מהפלט של ה-Agent שלך!)

### **טיפים ופתרון תקלות**

* לא קיבלת תשובה? בדוק View Output — איזה שדה עובר בין הנודים.
* בדוק לוגים (docker logs n8n)
* ודא Volumes משותפים (voice-temp, n8n\_data)
* כתובת NGROK חייבת להיות מעודכנת תמיד ב־WEBHOOK\_URL!
* אפשר להרחיב למסלולים נוספים (תמונה, מסמך, REST וכו')

## **פרק 10: סיכום, תחזוקה, טיפים, הרחבות ו-FAQ**

### **סיכום תהליך ההקמה**

בשלבים עד כה בנינו מערכת אוטומציה שלמה:

* התקנת סביבת עבודה (WSL/Ubuntu, Docker)
* התקנת Ollama והמודלים (AYA, mxbai-embed-large)
* התקנת ממשק ניהולי (Open WebUI)
* התקנת n8n, חיבור טלגרם, יצירת Workflow קול+טקסט
* התקנת whisper לתמלול אודיו
* חיבור NGROK לחשיפה מרחוק
* בדיקות, פתרון תקלות ושיפור

### **תחזוקה שוטפת**

1. **עדכון Docker/Ollama/n8n/whisper**

בדוק עדכונים:  
  
 bash  
CopyEdit  
sudo apt update && sudo apt upgrade

docker pull n8nio/n8n

docker pull ghcr.io/open-webui/open-webui:main

עדכן n8n:  
  
 bash  
CopyEdit  
npm install -g n8n

בדוק גרסאות Ollama:  
  
 bash  
CopyEdit  
ollama --version

1. **בדיקת Volumes וזיכרון**
   * וודא שהתיקיות (voice-temp, n8n\_data, דטה בייסים של RAG) לא מתמלאות מדי.
   * מחק קבצים ישנים/לא רלוונטיים.
2. **ניטור עומסים וביצועים**

פקודות לבדיקת עומס:  
  
 bash  
CopyEdit  
htop

docker stats

* + במקרה של עומס – שקול לשדרג חומרה או לפצל תהליכים.

1. **גיבוי**
   * גבה קבצי workflow, קבצי מודלים, קבצי תצורה, ו־volumes.
   * מומלץ לגבות ל־zip או git (לשימוש חוזר/שחזור מהיר).

### **טיפים להרחבה ושיפור**

* **ריבוי ערוצים:** ניתן להוסיף תמיכה בוואטסאפ, מייל, SMS, REST API, קלנדר ועוד — כל כלי ש־n8n תומך בו.
* **אוטומציות מותאמות:** שלח קבצים, תבצע OCR על מסמכים, תשלב חוקים/מחוללי דוחות.
* **שדרוג RAG:** אפשר להריץ FastAPI/שרת Flask מחוץ ל-n8n ולפנות אליו דרך HTTP Node (כבר בנינו תשתית לזה).
* **שליטה בהרשאות:** אפשר להגדיר אילו משתמשים/קבוצות מקבלים תשובות מהבוט.
* **התראות:** שלח התראות במייל/טלגרם במקרה של כשל במערכת.

### **שאלות נפוצות (FAQ)**

**Q:** *הבוט לא מגיב בטלגרם!* **A:** ודא שה־WEBHOOK\_URL בקובץ YAML נכון, בדוק שה־n8n רץ, ובדוק שה־Token עדכני.

**Q:** *אין תמלול לקול/הקלטה!* **A:** בדוק הרשאות ל־whisper (chmod +x ...), וודא שיש Volumes, בדוק קובץ transcribe.sh והאם ffmpeg/whisper עובדים בקונטיינר.

**Q:** *n8n לא עולה/נתקע/לא מגיב!* **A:**

* הרץ docker compose down && docker compose up -d
* בדוק פורטים (ss -tuln)
* בדוק לוגים (docker logs n8n)
* ודא שאין קונפליקט פורטים עם שירות אחר

**Q:** *שדרגתי משהו ועכשיו הכל נשבר!* **A:**

* חזור לגיבוי
* אשר שאין גרסאות לא תואמות (Node, Docker, Python)

**Q:** *איך להרחיב ל־RAG/חיבורים חיצוניים?* **A:** אפשר לבנות API FastAPI (ראה מדריך RAG), ולפנות אליו מ־n8n דרך HTTP Node.

### **הרחבות עתידיות**

* **RAG מתקדם:** אינטגרציה של RAG עם מסמכים דינמיים, תמיכה במסדי נתונים, REST, העלאת קבצים בממשק.
* **אבטחה:** הפרד את הגישה, הצפן Volumes, גבה את המערכת בקביעות.
* **בינה מלאכותית מתקדמת:** הוסף LLM/Embeddings נוספים, שפר את הפרומפטים.

### **שמירה על פרטיות**

כל המידע עובר אך ורק מקומית!  
 (חוץ מ־NGROK — אם לא מעוניינים בחשיפה, ניתן לשים פרוקסי/לעבוד ב־LAN בלבד.)

# **פרק 11: בניית RAG מקומי (Retrieval Augmented Generation) – שלב אחרי שלב**

### **מהי מערכת RAG?**

**RAG** היא מערכת חכמה שמאפשרת לשאול שאלות בשפה חופשית ולקבל תשובה מתוך מסמכים שהכנסת מראש, אפילו בעברית.

## **שלב 1: יצירת מבנה תיקיות לפרויקט**

1. פתח טרמינל במחשב שלך.
2. הרץ (כל שורה בנפרד):

bash

CopyEdit

mkdir -p ~/projects/rag-prototype/data

cd ~/projects/rag-prototype

* data/ – כאן נשים את כל המסמכים (PDF, DOCX, TXT, CSV).

## **שלב 2: יצירת קבצי הפרויקט**

תריץ:

bash

CopyEdit

touch rag\_pdf.py main.py app.py requirements.txt README.md

## **שלב 3: כתיבת requirements.txt**

העתק והדבק (פותח עם nano או vim):

txt

CopyEdit

langchain

langchain-ollama

langchain-chroma

langchain-community

chromadb

fastapi

uvicorn

pymupdf

python-docx

pandas

## **שלב 4: יצירת סקריפט לטעינת מסמכים (rag\_pdf.py)**

python

CopyEdit

from langchain\_community.document\_loaders import (

PyPDFLoader,

UnstructuredWordDocumentLoader,

TextLoader,

CSVLoader

)

from langchain\_community.embeddings import OllamaEmbeddings

from langchain\_chroma import Chroma

import os

DATA\_DIR = "data"

DB\_DIR = "./chrome\_langchain\_db"

embedding\_model = "mxbai-embed-large"

embeddings = OllamaEmbeddings(model=embedding\_model)

add\_documents = not os.path.exists(DB\_DIR)

all\_documents = []

for filename in os.listdir(DATA\_DIR):

path = os.path.join(DATA\_DIR, filename)

if filename.lower().endswith(".pdf"):

loader = PyPDFLoader(path)

elif filename.lower().endswith(".docx"):

loader = UnstructuredWordDocumentLoader(path)

elif filename.lower().endswith(".txt"):

loader = TextLoader(path, autodetect\_encoding=True)

elif filename.lower().endswith(".csv"):

loader = CSVLoader(file\_path=path)

else:

print(f"⛔ פורמט לא נתמך: {filename}")

continue

docs = loader.load()

all\_documents.extend(docs)

vector\_store = Chroma(

collection\_name="hebrew\_documents",

persist\_directory=DB\_DIR,

embedding\_function=embeddings

)

if add\_documents:

vector\_store.add\_documents(all\_documents)

print(f"✅ נוספו {len(all\_documents)} מסמכים למאגר.")

else:

print("ℹ️ המאגר כבר קיים. דלגנו על הוספה.")

## **שלב 5: כתיבת main.py — שאלות מהטרמינל**

python

CopyEdit

from langchain\_ollama.llms import OllamaLLM

from langchain\_core.prompts import ChatPromptTemplate

from langchain\_community.embeddings import OllamaEmbeddings

from langchain\_chroma import Chroma

model = OllamaLLM(model="aya")

embeddings = OllamaEmbeddings(model="mxbai-embed-large")

db\_location = "./chrome\_langchain\_db"

vector\_store = Chroma(

collection\_name="hebrew\_documents",

persist\_directory=db\_location,

embedding\_function=embeddings

)

retriever = vector\_store.as\_retriever()

template = """

המסמכים הרלוונטיים הם:

{reviews}

השאלה היא:

{question}

ענה תשובה ברורה ומבוססת רק על מה שיש במסמכים.

"""

prompt = ChatPromptTemplate.from\_template(template)

chain = prompt | model

while True:

print("\n" + "-"\*40)

question = input("✍️ שאל שאלה (או 'q' כדי לצאת): ")

if question.strip().lower() == "q":

break

reviews = retriever.invoke(question)

result = chain.invoke({"reviews": reviews, "question": question})

print("\n🤖 תשובה:", result)

## **שלב 6: כתיבת app.py – שרת FastAPI**

python

CopyEdit

from fastapi import FastAPI

from pydantic import BaseModel

from langchain\_ollama.llms import OllamaLLM

from langchain\_core.prompts import ChatPromptTemplate

from langchain\_community.embeddings import OllamaEmbeddings

from langchain\_chroma import Chroma

app = FastAPI()

model = OllamaLLM(model="aya")

embeddings = OllamaEmbeddings(model="mxbai-embed-large")

db\_location = "./chrome\_langchain\_db"

vector\_store = Chroma(

collection\_name="hebrew\_documents",

persist\_directory=db\_location,

embedding\_function=embeddings

)

retriever = vector\_store.as\_retriever()

template = """

המסמכים הרלוונטיים הם:

{reviews}

השאלה היא:

{question}

ענה תשובה ברורה ומבוססת רק על מה שיש במסמכים.

"""

prompt = ChatPromptTemplate.from\_template(template)

chain = prompt | model

class QuestionRequest(BaseModel):

question: str

@app.post("/ask")

async def ask\_question(req: QuestionRequest):

reviews = retriever.invoke(req.question)

result = chain.invoke({

"reviews": reviews,

"question": req.question

})

return {"answer": result}

## **שלב 7: הכנסת מסמכים**

* העתק קבצי PDF, DOCX, TXT, CSV אל תוך התיקיה data/
* דוג' פקודה:

bash

CopyEdit

cp /path/to/mydocs/\*.pdf ~/projects/rag-prototype/data/

## **שלב 8: יצירת סביבת עבודה והרצת התקנות**

1. פתח טרמינל בתיקיית הפרויקט:

bash

CopyEdit

cd ~/projects/rag-prototype

1. צור סביבה וירטואלית:

bash

CopyEdit

python3 -m venv venv

source venv/bin/activate

1. התקן תלויות:

bash

CopyEdit

pip install --upgrade pip

pip install -r requirements.txt

1. ודא ש־Ollama רץ ברקע (עם מודל aya וגם mxbai-embed-large!):

bash

CopyEdit

ollama pull aya

ollama pull mxbai-embed-large

ollama serve # אם צריך

## **שלב 9: טעינת המסמכים למערכת**

bash

CopyEdit

python rag\_pdf.py

* פעולה זו תיצור תיקיה בשם chrome\_langchain\_db ובה מאגר המסמכים (וקטורים).
* אם מעדכן מסמכים: מחק את התיקיה chrome\_langchain\_db לפני טעינה מחודשת.

## **שלב 10: בדיקת המערכת**

* בדיקת CLI:

bash

CopyEdit

python main.py

* בדיקת שרת API:

bash

CopyEdit

uvicorn app:app --reload --port 8000

* גש ל־<http://localhost:8000/docs>

## **שלב 11: דוגמה למבנה התיקיות לאחר ההרצה**

css

CopyEdit

rag-prototype/

├── app.py

├── chrome\_langchain\_db/

├── data/

│ ├── example.pdf

│ ├── mydoc.txt

│ └── ...

├── main.py

├── rag\_pdf.py

├── README.md

├── requirements.txt

└── venv/

### **סיכום**

* בנית מערכת RAG מלאה מקומית.
* הכנסת מסמכים – טעינתם – בדיקת שאילתות ב־CLI או API.
* אפשר עכשיו **לקשר את ה־RAG אל n8n** או לכל מערכת אחרת.

### **🔹 סיכום ביניים:**

* **כל המסמכים** נמצאים בתיקיית data/.
* **הפעלת rag\_pdf.py** יצרה מאגר אימבדינגים בתיקיית chrome\_langchain\_db/.
* **אפשר לשאול שאלות** דרך main.py (CLI) או app.py (API).

### **🔸 הסבר קצר על אימבדינגים (Embedding):**

**מה זה אימבדינג?**

* זה תהליך שבו המסמכים עוברים "המרה" לוקטורים מתמטיים (embedding vectors) כך שהמערכת תוכל למצוא מהם קטעים רלוונטיים לשאלה.
* קובץ האימבדינגים נשמר בתיקייה chrome\_langchain\_db — זו הספרייה שה־RAG ישלוף ממנה קטעים רלוונטיים כל פעם ששואלים שאלה.

**מתי צריך להריץ rag\_pdf.py?**

* רק כאשר הכנסת מסמכים חדשים, או מחקת/עדכנת קבצים קיימים בתיקיית data/.
* כל שינוי דורש להריץ שוב את rag\_pdf.py (ועדיף למחוק קודם את chrome\_langchain\_db).

## **פרק 12: חיבור n8n אל מערכת ה־RAG**

### **רקע:**

נחבר עכשיו את ה־n8n כך שכל הודעה מהבוט תועבר ל־API של RAG (שבנינו), ותחזור תשובה אוטומטית מתוך המסמכים.

#### **שלב 1: ודא שה־API של RAG רץ**

1. פתח טרמינל בתיקיית הפרויקט.
2. הפעל את FastAPI (חובה!):

bash

CopyEdit

source venv/bin/activate

uvicorn app:app --reload --port 8000

* דפדפן:<http://localhost:8000/docs>
* הנתיב שצריך: POST ל־/ask

#### **שלב 2: הכנת n8n לחיבור ל־RAG**

1. פתח את n8n (http://localhost:5678 או דרך כתובת ngrok).
2. הוסף Node חדש מסוג **Webhook** (POST).  
   * נתיב: /rag-agent (לדוג' /rag-agent)
3. אחרי Webhook, הוסף Node מסוג **HTTP Request**:  
   * **Method**: POST
   * **URL**: http://localhost:8000/ask
   * **Content-Type**: JSON

**Body/Parameters**:  
  
 json  
CopyEdit  
{

"question": "={{$json[\"question\"]}}"

}

* + *או תשתמש בשדה המתאים לפי איך שה־Webhook שלך בנוי.*

1. **חבר ביניהם**:  
    Webhook → HTTP Request
2. (אופציונלי) הוסף Node נוסף (Edit Fields או Set) כדי להמיר/לעבד את התשובה.
3. **סיום**:  
    הוסף Node "Telegram Send Message" להחזיר את התשובה לבוט.

#### **שלב 3: בדיקה**

* לחץ Execute Workflow על ה־Webhook.

שלח שאלה לבוט בטלגרם או עם curl:  
  
 bash  
CopyEdit  
curl -X POST http://localhost:5678/webhook-test/rag-agent \

-H "Content-Type: application/json" \

-d '{"question": "מה השם של הקיבוץ?"}'

* תראה תשובה מתוך המסמכים.

#### **דגשים חשובים:**

* **ה־API חייב לרוץ** (uvicorn app:app --reload --port 8000) בכל זמן שה־n8n משתמש ב־RAG!
* כתובת ה־HTTP צריכה להיות נכונה (localhost, או ה-IP שלך, או ngrok אם זה מחוץ ל־WSL).
* אפשר להרחיב — כל Node ב־n8n יכול לשלוח שאילתות ל־RAG.

### **סיכום:**

* הכנת RAG כולל אימבדינגים למסמכים
* הפעלת שרת FastAPI (app.py)
* חיבור n8n → Webhook → HTTP Request → RAG
* שאלות שנשלחות לבוט (טקסט/קול) יכולות להגיע למערכת RAG — לקבל תשובות **מהמסמכים שלך בעברית**!

### **איך זה בנוי ב־n8n:**

הסוכן החכם ב־n8n בנוי כ־**Workflow** שבו יש כמה "שלוחות" (Branches/Paths) — לדוגמה:

* שלוחה א': אם זו שאלה למסמכים → תפנה ל־RAG.
* שלוחה ב': אם זו בקשת אימייל → תשלח מייל.
* שלוחה ג': אם רוצים להוסיף פגישה לקלנדר → תפנה ל־Calendar.

## **🔹 שלב אחרי שלב — הוספת שלוחה ל־RAG בתוך n8n**

### **1. תכנן את ה־Switch**

הוסף Node מסוג **Switch** אחרי קליטת ההודעה:

* התנאי:  
  + אם ההודעה מכילה מילות מפתח שדורשות מסמכים (למשל: "מתוך המסמכים", "יש במסמך", "RAG", "פרוטוקול", "PDF" וכו').
  + אפשר גם סיווג לפי intent (מתקדם, בעזרת LLM).

### **2. הגדר שלוחה ל־RAG**

הוסף ב־Switch שלוחה (case) שנקראת "RAG" או "מסמכים".

* החיבור של branch זה יהיה:  
  + Edit Fields (למשוך טקסט רלוונטי מההודעה)
  + HTTP Request — בדיוק כמו שעשינו קודם:  
    - כתובת: http://localhost:8000/ask
    - שדה question: הטקסט של המשתמש

### **3. קבל תשובה והחזר אותה לבוט**

* קח את התשובה מה־HTTP Request (field בשם answer או איך שתחזיר).
* שלח אותה חזרה דרך **Telegram Send Message** (או דרך WhatsApp/email/כל ערוץ שבחרת).

### **דוגמה: תרשים זרימה כללי (ב־n8n)**

pgsql

CopyEdit

Telegram Trigger

│

Switch: איזה סוג שאלה?

│ │ │

"RAG" "Email" "Calendar"

│ │ │

HTTP Req Send Add to

(RAG) Email Calendar

│ │ │

Send Msg שלח אישור שלח קישור

### **דגשים:**

* **אם יש כמה סוגי משימות — כל שלוחה מקבלת Node משלה.**
* **אתה יכול להחליט בעצמך את ההיגיון:**
  + לפי מילות מפתח
  + לפי תפריט ("/rag", "/email", "/calendar")
  + או בעזרת LLM/סוכן ראשי שמחליט

### **דוגמת Node Switch (פשוטה):**

נניח, תנאי ל־RAG:

json

CopyEdit

{

"conditions": [

{

"operation": "contains",

"value1": "{{$json[\"text\"]}}",

"value2": "מסמך"

}

]

}

(אפשר להרחיב על עוד מילים, Regex, או אפילו סיווג ע"י LLM)

### **דוגמת HTTP Request ל־RAG:**

* **Method:** POST
* **URL:** http://localhost:8000/ask
* **Content-Type:** JSON

**Body:** json  
CopyEdit  
{

"question": "={{$json[\"text\"]}}"

}

### **סיכום**

* ככה בונים מערכת שבה כל **שאלה רלוונטית למסמכים** — נשלחת אוטומטית ל־RAG!
* כל שאר ה־branches של ה־n8n יכולים לבצע משימות נוספות (אימייל, וואטסאפ, אוטומציות, וכו').
* אפשר להרחיב בעתיד: להוסיף עוד סיווגים, Branchים, Agent חכם ראשי, אינטגרציות נוספות.

זו הדרך הכי פשוטה להתחיל — להשתמש **במילת קוד** (למשל "קוואבנגה") כטריגר שמפנה שלוחה אל RAG.

### **איך זה עובד בפועל?**

**המשתמש שולח לבוט הודעה:** CopyEdit  
קוואבנגה מה כתוב בסעיף 4 בפרוטוקול?

1. **Node ה־Switch ב־n8n** בודק אם הטקסט מתחיל או מכיל "קוואבנגה".  
   * אם כן — מעביר ל־Branch של RAG.
2. **ב־HTTP Request**:  
   * השאלה נשלפת מתוך ההודעה (אפשר להסיר אוטומטית את "קוואבנגה" — ב־Edit Fields/Function או Inline ב־JSON).
   * נשלחת ל־API של ה־RAG (/ask).
3. **התשובה מה־RAG** חוזרת ל־n8n, ומועברת ישירות חזרה לטלגרם/וואטסאפ וכו’.

#### **דוגמה ב־n8n:**

* **Switch**:

תנאי:  
  
 bash  
CopyEdit  
{{$json["text"].startsWith("קוואבנגה")}}

* **Edit Fields** (מנקה את המילה):

שדה חדש בשם "question":  
  
 bash  
CopyEdit  
{{$json["text"].replace("קוואבנגה", "").trim()}}

* **HTTP Request**:

שדה question:  
  
 bash  
CopyEdit  
{{ $json["question"] }}

* **Send Message**:

טקסט:  
  
 bash  
CopyEdit  
{{ $json["answer"] }}

### **טיפ:**

אתה יכול לשנות את מילת הקוד, להוסיף עוד מילים, או בהמשך לשלב עם מחלק (Classifier) אוטומטי שיזהה לבד האם זו שאלה למסמכים או משהו אחר.

#### **📂 מבנה תקיות:**

wasm

CopyEdit

~/rag-kibbutz-1/

data/ ← מסמכים של קיבוץ 1

chrome\_langchain\_db/ ← בסיס נתונים (נבנה אוט')

rag\_pdf.py

main.py

app.py (API: /ask1)

~/rag-kibbutz-2/

data/ ← מסמכים של קיבוץ 2

chrome\_langchain\_db/

rag\_pdf.py

main.py

app.py (API: /ask2)

# וכן הלאה...

#### **🧠 איך מחברים ב־n8n?**

* כל Branch ב־n8n מזהה את מילת הקוד (למשל: קוואבנגה1, קוואבנגה2, או שם קיבוץ).
* בהתאם — שולח את השאלה ל־HTTP Request שונה (לכל API של RAG נפרד: localhost:8001/ask1, localhost:8002/ask2, וכו’).
* מחזיר תשובה לפי המאגר הנכון.

#### **🎯 דוגמה לתנאי ב־Switch:**

* אם ההודעה מתחילה ב־"קוואבנגה1" → מפנה ל־/ask1
* אם ההודעה מתחילה ב־"קוואבנגה2" → מפנה ל־/ask2
* אפשר גם להוסיף מחלק חכם שיזהה לבד לפי הטקסט או מזהה אחר.

#### **🛠️ הרצה של כמה RAG במקביל:**

* כל API רץ על פורט אחר (8001, 8002, וכו').
* אין הגבלה אמיתית — רק וודא ש־Ollama רץ ויש לך מספיק זיכרון למודלים.

#### **🚀 יתרון:**

* כל קיבוץ/לקוח/צוות עובד עם המאגר שלו, פרטיות מלאה.
* אותו בוט — אבל הנתיב נקבע לפי מילת מפתח/Branch.