

בניית סוכן AI המותאם אישית

Building Custom AI Agents

מדריך Claude CLI מקיף

ספר בישול מעשי לפיתוח סוכנים חכמים



ד"ר סגל יורם

© Dr. Segal Yoram - כל הזכויות שמורות

2025-11-17

גרסה 1.0

2025 Dr. Segal Yoram □

כל הזכויות שמורות

נדפס בישראל

Printed in Israel

מהדורה ראשונה, נובמבר 2025

First Edition, November 2025

הצהרת רישיון

This work is provided for educational purposes.
All trademarks are property of their respective owners.
Claude CLI is a product of Anthropic PBC.

מוקדש לכל המפתחים החולמים לבנות סוכנים חכמים

שישנו את העולם

תקציר: Abstract

ספר זה מהווה מדריך מקיף ומעשי לבניית סוכני AI מותאמים אישית באמצעות Claude CLI. בעידן שבו בינה מלאכותית הופכת למרכזית בכל תחום, היכולת לפתח סוכנים אוטונומיים המותאמים לצרכים ספציפיים הפכה לכישור הכרחי. המדריך מציג גישה שיטתית הכוללת עשרה פרקים, החל מהתקנה ותצורה בסיסית, דרך הגדרת סוכנים ושילוב כלים, ועד לנושאים מתקדמים כמו RAG (Retrieval-Augmented Generation), אבטחה, ובדיקות. כל פרק בנוי כ"מתכון" מעשי עם דוגמאות קוד מלאות, המבוססות על סוכן דוגמה אחד המתפתח לאורך הספר: **יורם - סוכן ניהול הידע**. הספר מיועד לשלושה סוגי קוראים: מפתחים מנוסים המחפשים מדריך מהיר, חוקרים המעוניינים בהבנה תיאורטית עמוקה, וסטודנטים מתקדמים הרוצים ללמוד מאפס. כל פרק משלב תיאוריה ופרקטיקה, עם דגש על עקרונות אדריכליים ושיטות עבודה מומלצות.

מילות מפתח: AI Agents, Claude CLI, Natural Language Processing, Retrieval-Augmented Generation, Agentic Systems, Prompt Engineering, Tool Integration

Abstract

This book provides a comprehensive and practical guide to building custom AI agents using Claude CLI. In an era where artificial intelligence is becoming central to every domain, the ability to develop autonomous agents tailored to specific needs has become an essential skill.

The guide presents a systematic approach across ten chapters, starting from basic installation and configuration, through agent definition and tool integration, to advanced topics such as RAG (Retrieval-Augmented Generation), security, and testing. Each chapter is structured as a practical "recipe" with complete code examples, based on a single example agent that evolves throughout the book: **Yoram - the Knowledge Management Agent**.

The book targets three types of readers: experienced developers seeking a quick reference, researchers interested in deep theoretical understanding, and advanced students wanting to learn from scratch. Each chapter combines theory and practice, emphasizing architectural principles and best practices.

Keywords: AI Agents, Claude CLI, Natural Language Processing, Retrieval-Augmented Generation, Agentic Systems, Prompt Engineering, Tool Integration

הקדמה: Preface

ברוכים הבאים לעולם בניית סוכני ה-AI!

בשנים האחרונות, עברנו ממודלי שפה פסיביים שמשיבים לשאלות, לסוכנים אוטונומיים המסוגלים לתכנן, לבצע פעולות מורכבות, ואף ללמוד ולהשתפר עם הזמן. המעבר הזה אינו רק טכנולוגי - הוא פילוסופי: מכלי למענה על שאלות, לשותף המסוגל לחשוב, להחליט, ולפעול.

ספר זה נולד מתוך אמונה שכל מפתח, חוקר או סטודנט מתקדם יכול לבנות סוכנים חכמים - אם רק יינתנו לו הכלים והידע הנכונים. לאורך עשרה פרקים, תלמדו לא רק איך לבנות סוכנים, אלא גם מדוע לבנות אותם בדרך מסוימת, ומתי להשתמש בכל כלי וטכניקה. הגישה הייחודית של הספר מבוססת על "סוכן דוגמה חי" - יורם, סוכן ניהול ידע המתפתח איתכם מפרק לפרק. בפרק הראשון יורם הוא סוכן פשוט עם הגדרות בסיסיות. בפרק החמישי הוא כבר מסוגל ליצור דוחות מובנים. בפרק השביעי הוא משתמש ב-RAG לשליפת ידע ממסמכים, ובפרק התשיעי הוא עובר מבחני קבלה מקיפים. כך תוכלו לראות בזמן אמת איך סוכן הופך ממושג לכלי פונקציונלי ומתוחכם. אני מאחל לכם מסע למידה מרתק!

ד"ר סגל יורם

נובמבר 2025

תוכן העניינים

iii	1 *	
iii		תקציר
Abstract (English)	iv	
v	2 *	
v		הקדמה
ix		רשימת איורים
ix		רשימת טבלאות
iiix		רשימת קטעי קוד
1	I	חלק ראשון: יסודות
2	1	מבוא: עקרונות ה-Agentic
2	1.1	הבנת זרימת העבודה ה-Agentic
3	1.2	סביבת העבודה: Claude CLI
3	1.3	מבנה הספר וגישת הלמידה
5	1.4	הכירו: Yoram-Agent-Knowledge
6	1.5	סיכום ומבט קדימה
7	2	הגדרות ברמת המערכת והמשתמש
7	2.1	היררכיית קונפיגורציה
7	2.1.1	רמת המערכת (System-Level)
7	2.1.2	רמת המשתמש (User-Level)
7	2.1.3	רמת הפרויקט (Project-Level)
7	2.1.4	סדר עדיפות וירושה
8	2.2	התקנת Claude CLI
8	2.2.1	דרישות מקדימות
8	2.2.2	תהליך ההתקנה
9	2.2.3	התקנה מתקדמת: התקנה ממקור
9	2.3	הגדרת מפתח ה-API ברמת המשתמש
9	2.3.1	השגת מפתח API
9	2.3.2	אחסון מאובטח באמצעות משתני סביבה
01	2.3.3	שיקולי אבטחה

11	משתני סביבה	2.4
11	משתנים נפוצים	2.4.1
11	משתני סביבה לדיבאג	2.4.2
11	דוגמה משולבת: הקמת סביבה מלאה	2.4.3
21	ניהול מפתחות מרובים	2.4.4
31	סיכום	2.5
41	הגדרת פרויקט ואבני בניין: מבנה התיקיות הבסיסי	2
41	הסבר עקרוני: המבנה הקאנוני	2.1
51	ההגדרה המעשית: יצירת מבנה התיקיות	2.2
71	הדגמה: קובץ הקונפיגורציה הראשי	2.3
91	סיכום ומבט קדימה	2.4
12	השלד: בניית קובץ ה-Agent YAML	3
12	הסבר עקרוני: קובץ הקונפיגורציה (Agent Manifest)	3.1
12	הגדרות בסיסיות חובה	3.1.1
12	מבנה קובץ ה-YAML	3.1.2
22	ההגדרה המעשית: יצירת ה-YAML לסוכן ניהול הידע	3.2
22	שלב א': יצירת קובץ ה-YAML	3.2.1
22	שלב ב': הזנת תוכן השלד	3.2.2
32	דגשים על ההגדרות	3.2.3
32	אימות הקובץ	3.3
42	שימוש ב-Python לאימות YAML	3.3.1
52	יצירת מחולל YAML אוטומטי	3.4
03	שימוש במחולל	3.4.1
03	סיכום הפרק	3.5
13	חלק שני: כלים ושילוב	II
23	כלים מובנים	4
23	הסבר עקרוני: אימוץ כלים פנימיים	4.1
23	פילוסופיית הכלים: יכולות מול אחריות	4.1.1
23	קטגוריות הכלים המובנים	4.1.2
33	שלושת הכלים החיוניים: read, websearch, grep	4.1.3
33	ההגדרה המעשית: הוספת הכלים לקובץ ה-YAML	4.2
43	מבנה רשימת הכלים	4.2.1
53	הבנת השינויים	4.2.2
53	דוגמה לשימוש מעשי	4.2.3
63	הרחבה עתידית: כלים נוספים	4.2.4

73	עקרון ההרשאה המינימלית (Principle of Least Privilege)	4.3
73	סיכוני הרשאות מופרזות	4.3.1
73	מטריצת כלים לפי סוגי סוכנים	4.3.2
73	אסטרטגיית הוספה הדרגתית	4.3.3
83	אימות והרצה: בדיקת הסוכן המורחב	4.4
83	בדיקת תקינות הגדרה	4.4.1
93	בדיקת כלי read	4.4.2
93	בדיקת כלי websearch	4.4.3
93	בדיקת כלי grep	4.4.4
04	פתרון בעיות נפוצות	4.5
04	בעיה: הסוכן אינו משתמש בכלים	4.5.1
04	בעיה: שגיאת הרשאות בעת שימוש ב-read	4.5.2
14	בעיה: websearch לא מחזיר תוצאות רלוונטיות	4.5.3
14	סיכום	4.6
24	יכולות מותאמות אישית: בניית SKILL	4.7
24	הסבר עקרוני: הצורך ב-SKILL	4.7.1
34	ההגדרה המעשית: יצירת תיקיית ה-SKILL וקובץ ה-SKILL.md	4.7.2

94	6 אינטגרציה חזקה: הוספת קוד Python	
94	6.1	הסבר עקרוני: הרחבת ה-SKILL באמצעות קוד
94	6.1.1	מבנה ה-Skill המשולב
05	6.2	ההגדרה המעשית: יצירת התשתית
05	6.2.1	ארגון תיקיית ה-scripts
05	6.3	עיבוד נתונים וניתוח: סקריפט ה-Python
15	6.3.1	הקוד המלא של data_processor.py
45	6.4	זרימת העבודה של אינטגרציה Python-SKILL
45	6.4.1	עדכון SKILL.md
55	6.4.2	חיבור ה-SKILL לסוכן
65	6.5	הוספת ה-data_processor.py ל-structured-report
75	6.6	סיכום והמשך

III חלק שלישי: תכונות מתקדמות 85

95	זיכרון וקטורי (RAG): ניהול ידע מתקדם	7
95	הסבר עקרוני: חריגה מגבולות ההקשר	7.1
06	עקרונות Retrieval-Augmented Generation	7.2
06	שלב האינדקס: הכנת מאגר הידע	7.2.1
06	שלב השאילתה: הבנת כוונת המשתמש	7.2.2
16	שלב האחזור: מציאת המידע הרלוונטי	7.2.3

16	שלב הייצור: יצירת תשובה מושכלת	7.2.4
16	מודלי הטמעה: nomic-embed-text	7.2.5
26	הגדרת vector_memory	7.3
26	התקנת Qdrant באמצעות Docker	7.3.1
36	יצירת Collection: הגדרת מבנה המאגר	7.3.2
36	אינדקס מסמכים: הוספת תוכן למאגר	7.3.3
56	תצורת YAML מלאה: חיבור הסוכן למאגר	7.3.4
66	הגדרת knowledge Retriever – פרוטוקול MCP	7.4
66	מהו Model Context Protocol?	7.4.1
76	מבנה שרת MCP לאחזור ידע	7.4.2
07	אינטגרציה עם הסוכן: תצורת MCP	7.4.3
17	הוספת RAG לסוכן Yoram-Agent-Knowledge	7.5
17	תצורת Yoram ללא RAG (מפרק שישי)	7.5.1
27	תצורת Yoram עם RAG מלא	7.5.2
57	פרומפט מערכת מתקדם: שילוב RAG בתהליך החשיבה	7.5.3
57	דוגמה מעשית: Yoram בפעולה עם RAG	7.5.4
67	סיכום ומבט קדימה	7.6
87	הסבר עקרוני: סיכונים ובקרה על פקודות מערכת	7.7
87	העוצמה והסכנה של כלי ה-Bash	7.7.1
97	עיקרון ההרשאה המינימלית: Principle of Least Privilege	7.7.2
97	bashoutput לעומת bash: שתי רמות בקרה	7.7.3
97	תרחישי סיכון וצעדי מניעה	7.7.4
18	ההגדרה המעשית: הגבלת הרשאות ב-YAML	7.8
18	עדכון קובץ ה-YAML: הוספת כלי Bash מבוקרים	7.8.1
38	ניתוח ההגדרות: שכבות האבטחה	7.8.2
38	דוגמאות שימוש: הגוון בין בטוח למסוכן	7.8.3
48	מבחן העקרונות: אימות התנהגות	7.8.4
58	מגבלות והסתייגויות	7.8.5
58	סיכום ומסקנות	7.8.6
68	נספח: רשימת פקודות מסוכנות לחסימה	7.8.7
78	9 הרצה ובדיקה	
78	הפעלת הסוכן: הפקודה claude agent run	9.1
78	תחביר הפקודה הבסיסית	9.1.1
88	פרמטרים נוספים ומתקדמים	9.1.2
88	בדיקת כלים מובנים	9.2
98	בדיקת כלי read	9.2.1
98	בדיקת כלי websearch	9.2.2
98	בדיקת כלי grep	9.2.3

19	בדיקת SKILLs מותאמים	9.3
19	9.3.1 הכנת נתוני בדיקה	
29	9.3.2 הפעלת ה-SKILL עם נתונים אמיתיים	
29	9.3.3 בדיקת טיפול בשגיאות	
29	9.4 אימות שליפת RAG	
29	9.4.1 הכנת סביבת הבדיקה	
29	9.4.2 בדיקת שליפת מידע מהזיכרון הווקטורי	
29	9.4.3 בדיקת שאילתות מורכבות	
79	9.5 הדגמת תהליך עבודה מלא End-to-End	
79	9.5.1 תרחיש הדגמה: ניתוח מקיף של פרויקט	
79	9.5.2 ניתוח התהליך	
99	9.6 פתרון בעיות נפוצות	
99	9.6.1 שגיאה: חיבור למסד הנתונים הווקטורי נכשל	
001	9.6.2 שגיאה: SKILL לא נמצא	
001	9.6.3 שגיאה: מודל לא זמין	
101	9.6.4 בעיה: הסוכן איטי מדי	
101	9.7 סיכום והמשך הדרך	
201	9.7.1 מה למדנו?	
201	9.7.2 לאן מכאן?	
301	9.7.3 מחשבות סיום	
	9.8 English References	104
501	10 רקע והשוואה ראשונית	
501	10.1 השוואה ראשונית: YAML לעומת Markdown	
601	11 השוואה מפורטת בין שלושת הפורמטים	
601	11.1 טבלת השוואה כוללת	
701	12 דוגמאות קוד מעשיות	
701	12.1 דוגמה ב-YAML	
701	12.2 דוגמה ב-JSON	
801	12.3 דוגמה ב-Markdown	
011	13 השוואה כללית והמלצות שימוש	
011	13.1 מתי להשתמש בכל פורמט?	
011	13.2 שילוב הפורמטים בפרויקט Claude CLI	
111	13.3 סיכום ומסקנות	

רשימת האיוורים

1	זרימת עבודה Agentic: מחזור איטרטיבי של תכנון, ביצוע כלים, וסינתזה .	2
2	מבנה הספר: התקדמות הדרגתית מסוכן בסיסי לסוכן מתקדם עם RAG	
4	ואבטחה	

רשימת הטבלאות

71	Knowledge-Agent-Yoram	מבנה תיקיות הפרויקט	1
83	Recommended Tool Matrix by Agent Role		2
08	Agentic	תרחישי סיכון וצעדי הגנה במערכת	3
68		פקודות מסוכנות ודרכי המניעה	4
501	Markdown	השוואה ראשונית: YAML לעומת	5
601	Markdown ,JSON ,YAML	השוואה מפורטת:	6
011		המלצות שימוש לפי מקרה שימוש	7

רשימת קטעי קוד

12	מבנה בסיסי של קובץ LMay tnegA	1
22	יצירת קובץ lmay.tsylana-trepxe	2
22	קובץ lmay.tsylana-trepxe המלא	3
42	סקריפט אימות קובץ LMay	4
62	מחולל קובצי LMay tnegA אוטומטי	5
03	הרצת מחולל LMay במצב אינטראקטיבי	6
05	יצירת מבנה תיקיות ל-SKILL עם Python	7
15	data_processor.py – סקריפט עיבוד נתונים (130 שורות)	8
45	עדכון קובץ SKILL.md עם הפניה ל-Python	9
55	עדכון הסוכן להכיל את ה-SKILL	01
26	הרצת Qdrant עם Docker	11
36	יצירת collection ב-Qdrant	21
36	אינדקס מסמכים ב-Qdrant	31
56	תצורה מלאה של vector_memory	41
76	שרת MCP לאחזור ידע מ-Qdrant	51
07	אינטגרציה של שרת MCP בתצורת הסוכן	61
17	תצורת Yoram המקורית (ללא RAG)	71
27	תצורת Yoram המשודרגת (עם RAG)	81

חלק I

חלק ראשון: יסודות

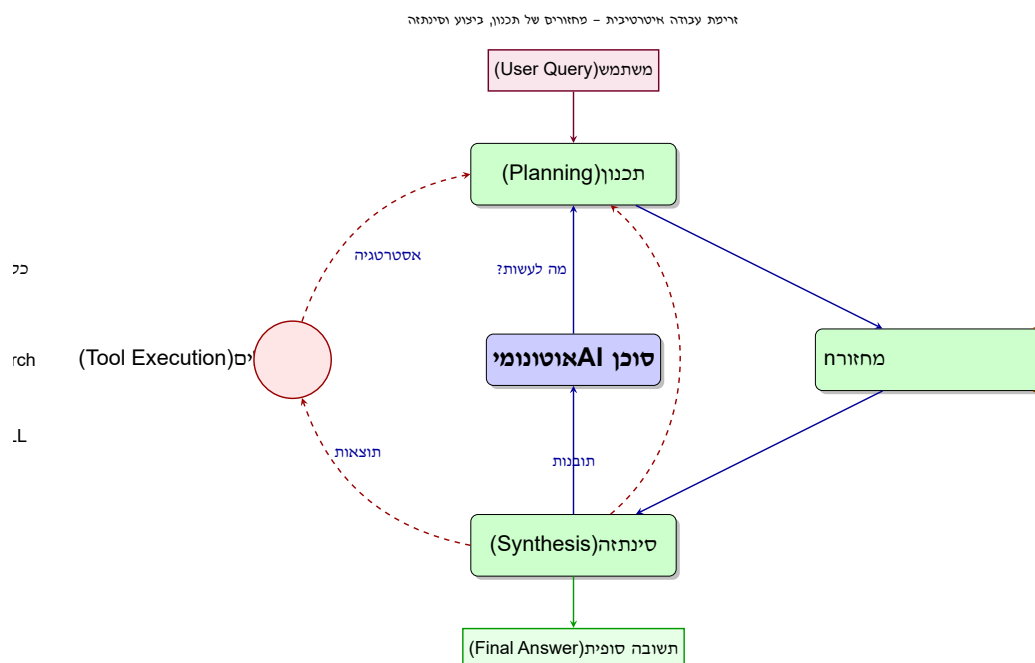
1 מבוא: עקרונות ה-Agentic

הדור הנוכחי של מודלי שפה גדולים מהווה קפיצת מדרגה משמעותית ביכולת של מערכות בינה מלאכותית לעבד טקסט, לנתח מידע ולייצר תוכן. אולם, עבור רבים מאיתנו – מהנדסי תוכנה, חוקרים וסטודנטים מתקדמים – קיים פער בין הפוטנציאל העצום של מודלים אלו לבין היכולת המעשית שלנו לרתום אותם למשימות ממשיות ומורכבות. הספר שלפניכם נועד לגשר על פער זה באמצעות גישה מובנית, מעשית ומעמיקה.

1.1 הבנת זרימת העבודה ה-Agentic

כאשר אנו משוחחים עם מודל AI באופן ישיר – למשל, באמצעות ממשק צ'אט רגיל – אנו מקבלים תשובה מידית על בסיס ההקשר שסיפקנו באותו רגע. זהו מודל עבודה **תגובתי** (Reactive): אנו שואלים, המודל עונה, והאינטראקציה מסתיימת. גישה זו מתאימה לשאלות פשוטות, אך היא מוגבלת ביכולתה לטפל במשימות מורכבות הדורשות גישה למידע חיצוני, ביצוע פעולות רב-שלביות או שמירה על הקשר לאורך זמן.

לעומת זאת, **סוכן AI** (AI Agent) פועל בצורה **יזומה ואוטונומית**. סוכן אינו רק מגיב לשאלה בודדת, אלא מסוגל לתכנן רצף פעולות, להפעיל כלים חיצוניים, לאסוף מידע ממקורות מגוונים ולחזור על תהליך זה עד להשגת המטרה [1]. כפי שמתואר באיור 1, זרימת העבודה ה-Agentic מורכבת ממחזורים של תכנון, ביצוע ואימות, כאשר בכל שלב הסוכן מעריך את התקדמותו ומתאים את אסטרטגיתו בהתאם.



איור 1: זרימת עבודה Agentic: מחזור איטרטיבי של תכנון, ביצוע כלים, וסינתזה

למהנדס התוכנה המנוסה, הרעיון דומה לארכיטקטורת microservices או לתזמון orchestration של משימות מורכבות: מודל ה-AI משמש כשכבת ההחלטה המרכזית, בעוד הכלים

והממשקים החיצוניים מספקים את היכולות הפרוצדורליות. עבור החוקר האקדמי, מדובר במימוש של **תורת הסוכנים** (Agent Theory) [2], [3] – ישות אוטונומית בעלת מטרות, תפיסת סביבה ויכולת פעולה. ועבור הסטודנט המתקדם, זוהי הזדמנות להבין כיצד לקחת מודל שפה גדול ולהפוך אותו מכלי תגובתי לישות פעילה ויעילה.

1.2 סביבת העבודה: Claude CLI

Claude CLI הוא כלי שורת פקודה רשמי המאפשר אינטראקציה מתקדמת עם מודלים של Anthropic [4], [5]. בניגוד לממשק הצ'אט הגרפי, ה-CLI מספק שליטה מלאה על הקשר השיחה, על הכלים הזמינים לסוכן, ועל מבנה הפרויקט כולו. זוהי הסביבה האידיאלית לפיתוח סוכנים מותאמים אישית.

למה CLI ולא ממשק גרפי? התשובה טמונה ב**אוטומציה ובשחזוריות**. כאשר אנו מגדירים סוכן באמצעות קבצי קונפיגורציה (YAML, Markdown), אנו יוצרים תיעוד חי של התנהגותו. כל שינוי נשמר בקובץ, ניתן לבקרה בגרסאות, ואפשר לשותף את הסוכן עם צוות או לשכפל אותו לסביבות שונות. זוהי גישה **מקצועית** המתאימה לסביבות ייצור ולפיתוח ארגוני. Claude CLI תומך בארכיטקטורה היררכית של הגדרות: רמת המערכת, רמת המשתמש ורמת הפרויקט. הבנת ההיררכיה הזו חיונית לשליטה מלאה על התנהגות הסוכן, ופרק 1 יעסוק בכך לעומק. בשלב זה, חשוב להבין שה-CLI אינו רק כלי לשיחה, אלא **פלטפורמת פיתוח** שלמה לבניית סוכנים אינטליגנטיים.

1.3 מבנה הספר וגישת הלמידה

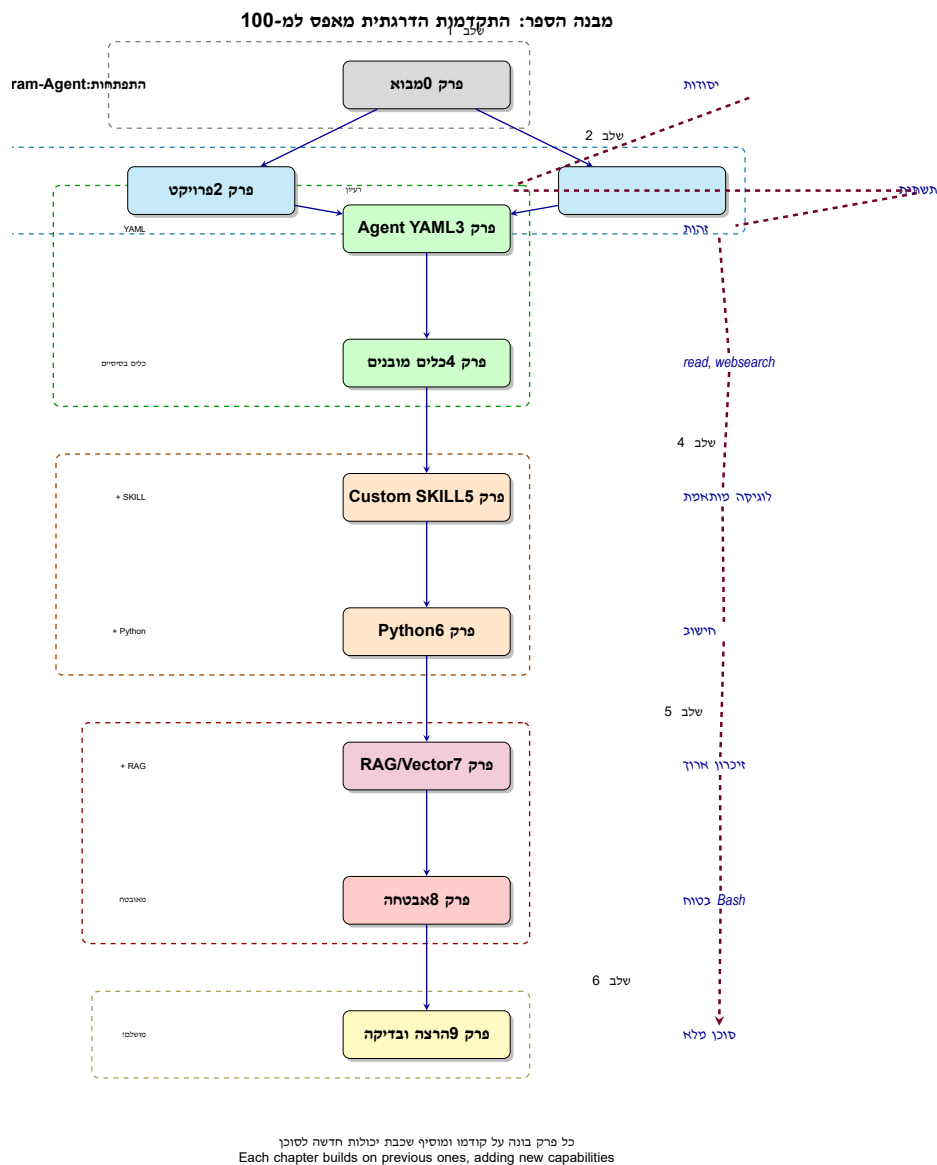
ספר זה בנוי בסגנון "**ספר בישול**" – גישה פרוגרסיבית מאפס למ-100. כל פרק מתמקד במרכיב מפתח אחד בבניית סוכן, מציג את העקרונות התיאורטיים, מדגים אותם בפועל ומוסיף שכבה נוספת ליכולות הסוכן שלנו. בסוף המסע, נבנה יחד סוכן מלא ופונקציונלי שמשלב כלים מובנים, יכולות מותאמות אישית, קוד Python, זיכרון וקטורי (RAG) ומדיניות אבטחה מחמירה.

איור 2 מציג את מבנה הפרקים ואת ההתקדמות ההדרגתית של הסוכן שלנו. כל פרק בנוי על קודמו, וכל שלב מוסיף יכולת חדשה שמרחיבה את הפוטנציאל של הסוכן. **גישת שלוש הפרסונות:** ספר זה נכתב תוך מודעות לשלושה קהלים עיקריים:

1. **מהנדס תוכנה בכיר:** אתם מחפשים דרכים מהירות ליישום, best practices ודוגמאות קוד עובדות. הספר יספק לכם נתיבים ישירים להשגת תוצאות, עם דגש על ארכיטקטורה נכונה ועל אינטגרציה עם קוד קיים.

2. **חוקר/אקדמאי:** אתם מעוניינים בעומק תיאורטי, בהבנת העקרונות הבסיסיים מאחורי הטכנולוגיה ובהשוואה בין גישות שונות. הספר יכלול ביבליוגרפיה, הפניות למקורות אקדמיים וניתוח של ארכיטקטורת הסוכנים ושל טכנולוגיית RAG.

3. **סטודנט מתקדם/מתחיל:** אתם זקוקים להסברים ברורים, לצעדים מפורטים ולתובנות יסודיות. הספר ילווה אתכם בכל שלב, יסביר מושגים בסיסיים לפני מעבר לנושאים מתקדמים, ויציין מלכודות נפוצות ואיך להימנע מהן.



איור 2: מבנה הספר: התקדמות הדרגתית מסוכן בסיסי לסוכן מתקדם עם RAG ואבטחה

סגנון הכתיבה: בעקבות גישתו של יובל נח הררי בספריו, נעדיף כאן **פרוזה זורמת** על פני רשימות נקודות יבשות. נספר סיפור על איך סוכנים מתפתחים, על השיקולים שמובילים להחלטות ארכיטקטוניות ועל הקשר הרחב יותר של בינה מלאכותית אוטונומית. זה לא רק מדריך טכני – זה מסע רעיוני.

1.4 הכירו: Yoram-Agent-Knowledge

בליבו של ספר זה עומד **סוכן דוגמה מתפתח** שנקרא Yoram-Agent-Knowledge (בקצרה: expert-analyst). זהו סוכן ניהול ידע מומחה, שתפקידו לשלוף מידע מתקדם, לנתח מסמכים טכניים ולסכם ידע מקיף מתוך מקורות מגוונים. הסוכן הזה ילווה אותנו לאורך כל הפרקים, ובכל שלב נוסיף לו יכולת חדשה:

- **פרק 1-2:** נקים את התשתית הכללית ואת מבנה התיקיות הפורמלי של הפרויקט.
- **פרק 3:** נגדיר את **הזהות** של הסוכן – שמו, תיאורו, המודל שבו הוא ישתמש (Opus) והנחיות המערכת שלו (System Prompt).
- **פרק 4:** נצייד אותו ב**כלים מובנים** כגון read, websearch, grep – היכולת לקרוא קבצים, לחפש באינטרנט ולסרוק תוכן.
- **פרק 5:** נבנה **יכולת מותאמת אישית** (SKILL) בשם structured-report, שתאפשר לו לעבד נתונים גולמיים ולהפיק דוחות מובנים.
- **פרק 6:** נשלב **קוד Python** בתוך ה-SKILL, ונאפשר לסוכן לבצע חישובים סטטיסטיים וניתוח נתונים מתקדם.
- **פרק 7:** נוסיף **זיכרון וקטורי** (RAG – Retrieval-Augmented Generation) [6], שיאפשר לסוכן לגשת למאגרי ידע פנימיים גדולים שחורגים מחלון ההקשר של המודל.
- **פרק 8:** נטמיע **מדיניות אבטחה** מחמירה לשליטה על כלי ה-Bash, ונוודא שהסוכן פועל בהתאם לעקרון ההרשאה המינימלית (Principle of Least Privilege).
- **פרק 9:** נפעיל את הסוכן המלא, נבצע בדיקות end-to-end ונוודא שכל הרכיבים משולבים ופועלים כצפוי.

מדוע בחרנו בשם "Yoram"? בהתאם להנחיות הפרויקט, שימוש בשם זה מחייב **טון מקצועי ורשמי**. הסוכן שלנו אינו עוזר אישי קליל, אלא אנליסט בכיר שמתחייב לדיוק, לשיטתיות ולמקצועיות מוחלטת. זהו סוכן שמיועד לסביבות עבודה ארגוניות, למחקר אקדמי ולמשימות ניתוח מורכבות.

בכל פרק נתעד את השינויים המדויקים שנבצע בקובץ הקונפיגורציה של הסוכן (expert-analyst.yaml), נסביר את השיקולים שהובילו לבחירות שלנו, ונדגים את השפעת כל תוספת על התנהגותו של הסוכן. בסוף המסע, תהיה לכם הבנה מלאה של כל מרכיב בסוכן, ותוכלו ליצור סוכנים משלכם המותאמים לצרכים הספציפיים שלכם.

1.5 סיכום ומבט קדימה

פרק זה הניח את התשתית המושגית להבנת סוכני AI: הבחנו בין גישה תגובתית לבין זרימת עבודה Agentic, הכרנו את סביבת Claude CLI כפלטפורמת פיתוח מקצועית, הבנו את מבנה הספר וגישת הלמידה ההדרגתית שלו, והצגנו את הסוכן המתפתח שילווה אותנו – Yoram-Agent-Knowledge.

בפרק הבא, נצא לדרך המעשית: נתקין את Claude CLI, נגדיר את מפתח ה-API שלנו באופן מאובטח, ונבין את ההיררכיה של הגדרות המערכת. זהו הבסיס הטכני שעליו נבנה את כל המבנה.

אם אתם מהנדסים – התכווננו לעבודה מובנית ומתועדת היטב. אם אתם חוקרים – תמצאו כאן תובנות תיאורטיות ומקורות מוסמכים. ואם אתם סטודנטים – אל תדאגו, נלך יחד צעד אחר צעד.

בואו נתחיל לבנות.

2 הגדרות ברמת המערכת והמשתמש

לפני שניתן לנצל את מלוא הפוטנציאל של Claude CLI, עלינו להבין את העקרונות הארכיטקטוניים המנחים את מערכת ההגדרות שלו. בפרק זה נבחן את היררכיית הקונפיגורציה, נתקין את הכלי, ונגדיר את פרמטרי האבטחה הבסיסיים הנדרשים לתקשורת עם API של Anthropic. הבנת המבנה ההיררכי אינה רק עניין טכני - היא משקפת פילוסופיה תכנונית המאפשרת גמישות מקסימלית תוך שמירה על ברירות מחדל סבירות.

2.1 היררכיית קונפיגורציה

מערכת Claude CLI מאמצת מודל היררכי תלת-רמתי המאפשר הפרדה ברורה בין הגדרות גלובליות, הגדרות ברמת המשתמש, והגדרות ברמת פרויקט ספציפי. עקרון זה, הלקוח מתחום ניהול התצורה ב-Ubuntu, מבטיח שכל רמה יכולה לדרוס את הרמה הכללית יותר ממנה.

2.1.1 רמת המערכת (System-Level)

ברמת המערכת, ההגדרות חלות על כל המשתמשים במחשב. במערכות Unix/Linux, קבצי קונפיגורציה כגון `/etc/claude/config` עשויים להכיל הגדרות ארגוניות כלל-מערכתיות. רמה זו רלוונטית במיוחד בסביבות ארגוניות שבהן מנהל מערכת מעוניין לאכוף מדיניות אבטחה או הגדרות proxy אחידות.

2.1.2 רמת המשתמש (User-Level)

רמת המשתמש מהווה את נקודת המוצא המעשית לרוב המפתחים. הגדרות אלו מאוחסנות בתיקיית הבית של המשתמש, בדרך כלל תחת `~/.claude/` או באמצעות משתני סביבה ב-`~/.bashrc` או `~/.zshrc`. ההפרדה בין רמת המשתמש לרמת המערכת מאפשרת למפתח להתאים אישית את סביבת העבודה שלו מבלי להשפיע על משתמשים אחרים.

2.1.3 רמת הפרויקט (Project-Level)

רמת הפרויקט מספקת את הדיוק הגבוה ביותר. קבצים כגון `.claude/config.json` בשורש הפרויקט מאפשרים הגדרות ספציפיות לפרויקט, כגון בחירת מודל AI מסוים, הגדרת פרמטרי `temperature`, או שילוב עם כלי CI/CD ספציפיים. עקרון זה חיוני בעת עבודה על מספר פרויקטים בו-זמנית, כאשר לכל פרויקט דרישות שונות.

2.1.4 סדר עדיפות וירושה

מנגנון העדיפות פועל מהספציפי לכללי: הגדרות ברמת הפרויקט גוברות על הגדרות ברמת המשתמש, אשר בתורן גוברות על הגדרות ברמת המערכת. ניתן לבטא זאת באמצעות יחס העדיפות:

מערכת > משתמש > פרויקט

מנגנון זה מאפשר למשל להגדיר מפתח API כללי ברמת המשתמש, אך לעקוף אותו עבור פרויקט ספציפי הדורש חשבון ארגוני שונה.

2.2 התקנת Claude CLI

התקנת Claude CLI היא תהליך פשוט המתבסס על מנהל החבילות pip של Python. נדגים את התהליך באמצעות הקמת פרויקט לדוגמה בשם Knowledge-Agent-Yoram, פרויקט המדמה סוכן מחקר אקדמי.

2.2.1 זרישות מקדימות

לפני ההתקנה, יש לוודא שמותקנת גרסת Python מספקת (מומלץ Python 3.8 ומעלה) ומנהל החבילות pip. ניתן לאמת זאת באמצעות:

```
# Verify Python version
python --version
# Expected output: Python 3.8.x or higher

# Verify pip installation
pip --version
```

2.2.2 תהליך ההתקנה

ההתקנה מתבצעת באמצעות פקודה בודדת. מומלץ להשתמש בסביבה וירטואלית (virtual environment) כדי לבודד את התלויות:

```
# Create project directory
mkdir -p ~/projects/Knowledge-Agent-Yoram
cd ~/projects/Knowledge-Agent-Yoram

# Create and activate virtual environment
python -m venv venv
source venv/bin/activate # On Windows: venv\Scripts\activate

# Install Claude CLI
pip install claude-cli

# Verify installation
```

```
claude --version
```

פקודת `claude --version` אמורה להחזיר את מספר הגרסה הנוכחית, מה שמאשר שההתקנה הושלמה בהצלחה. במקרה של כשל, יש לבדוק את לוגים באמצעות `pip install --verbose claude-cli`.

2.2.3 התקנה מתקדמת: התקנה ממקור

למפתחים המעוניינים לתרום לפרויקט או לנסות תכונות ניסיוניות, ניתן להתקין ישירות ממאגר GitHub:

```
# Clone repository
git clone https://github.com/anthropics/claude-cli.git
cd claude-cli
```

```
# Install in development mode
pip install -e .
```

הדגל `-e` (editable mode) מאפשר שינויים בקוד המקור להשתקף מיד מבלי צורך בהתקנה מחדש.

2.3 הגדרת מפתח ה-API ברמת המשתמש

אימות מול שירותי Anthropic מתבצע באמצעות מפתח API ייחודי. מפתח זה מהווה אישור גישה אישי ויש להגן עליו בקפדנות. נבחן את האסטרטגיות המומלצות לאחסון ושימוש במפתח.

2.3.1 השגת מפתח API

ראשית, יש להירשם לשירות Anthropic דרך <https://console.anthropic.com>. לאחר יצירת חשבון, ניתן ליצור מפתח API חדש תחת לשונית API Keys. כל מפתח מתחיל בקידומת `sk-ant-` ומכיל סדרת תווים אלפאנומריים.

2.3.2 אחסון מאובטח באמצעות משתני סביבה

השיטה המומלצת לאחסון מפתח ה-API היא באמצעות משתני סביבה. בניגוד לאחסון בקבצי קונפיגורציה, משתני סביבה אינם מסתכנים בחשיפה מקרית דרך מערכות ניהול גרסאות כמו Git. עבור משתמשי bash:

```
# Edit ~/.bashrc
nano ~/.bashrc
```

Add the following line at the end:

```
export ANTHROPIC_API_KEY="sk-ant-your-actual-api-key-here"
```

```
# Reload the configuration
source ~/.bashrc
```

עבור משתמשי zsh (ברירת המחדל ב-macOS):

```
# Edit ~/.zshrc
nano ~/.zshrc
```

```
# Add the following line at the end:
export ANTHROPIC_API_KEY="sk-ant-your-actual-api-key-here"
```

```
# Reload the configuration
source ~/.zshrc
```

לאימות שהמפתח נטען כראוי:

```
# Verify environment variable is set
# (shows first 10 characters only)
echo ${ANTHROPIC_API_KEY:0:10}
# Expected output: sk-ant-api

# Test connection
claude auth status
```

2.3.3 שיקולי אבטחה

מספר עקרונות אבטחה חיוניים:

הפרדת סודות מקוד: אף פעם אל תשמרו מפתחות API בקבצים הנמצאים תחת בקרת גרסאות. קובץ `.gitignore`. חייב לכלול את כל קבצי הקונפיגורציה המקומיים.

הרשאות קבצים: יש לוודא שקבצי `.bashrc` או `.zshrc`. נגישים לקריאה רק למשתמש הנוכחי:

```
# Set proper permissions (owner read/write only)
chmod 600 ~/.bashrc
chmod 600 ~/.zshrc
```

רוטציה תקופתית: מומלץ להחליף מפתחות API מעת לעת, במיוחד אם קיים חשש לחשיפה.

ניטור שימוש: לוח הבקרה של Anthropic מציג סטטיסטיקות שימוש. פעילות חריגה עשויה להצביע על שימוש לא מורשה במפתח.

2.4 משתני סביבה

מעבר למפתח ה-API הבסיסי, Claude CLI תומך במשתני סביבה נוספים המאפשרים כיוונון עדין של ההתנהגות.

2.4.1 משתנים נפוצים

ANTHROPIC_BASE_URL: מאפשר להפנות את הבקשות לשרת חלופי, שימושי בסביבות פיתוח או עבור proxy ארגוני:

```
export ANTHROPIC_BASE_URL="https://api-proxy.company.com"
```

CLAUDE_DEFAULT_MODEL: מגדיר את מודל ברירת המחדל לשימוש (כגון `claude-sonnet-4-5-20250929` או `claude-opus-4-20250514`):

```
export CLAUDE_DEFAULT_MODEL="claude-sonnet-4-5-20250929"
```

CLAUDE_CONFIG_DIR: מאפשר לשנות את מיקום תיקיית הקונפיגורציה מברירת המחדל `~/.claude`:

```
export CLAUDE_CONFIG_DIR="$HOME/.config/claude"
```

HTTP_PROXY ו-**HTTPS_PROXY**: עבור סביבות תאגידיות הדורשות proxy:

```
export HTTP_PROXY="http://proxy.company.com:8080"
```

```
export HTTPS_PROXY="https://proxy.company.com:8443"
```

2.4.2 משתני סביבה לדיבאג

לצורכי ניפוי באגים ופיתוח:

```
# Enable verbose logging
```

```
export CLAUDE_DEBUG=1
```

```
# Set log level (DEBUG, INFO, WARNING, ERROR)
```

```
export CLAUDE_LOG_LEVEL="DEBUG"
```

```
# Specify custom log file
```

```
export CLAUDE_LOG_FILE="$HOME/logs/claude-debug.log"
```

משתנים אלו מפעילים רישום מפורט של כל בקשה ותגובה, שימושי לאבחון בעיות תקשורת או התנהגות בלתי צפויה.

2.4.3 דוגמה משולבת: הקמת סביבה מלאה

נדגים הגדרה מלאה של סביבת עבודה לפרויקט Knowledge-Agent-Yoram, כולל כל משתני הסביבה הרלוונטיים:

```

# Create comprehensive environment setup
cat >> ~/.zshrc << 'EOF'

# === Claude CLI Configuration ===
# API Authentication
export ANTHROPIC_API_KEY="sk-ant-your-actual-key-here"

# Default model selection
export CLAUDE_DEFAULT_MODEL="claude-sonnet-4-5-20250929"

# Custom config directory
export CLAUDE_CONFIG_DIR="$HOME/.config/claude"

# Proxy settings (if needed)
# export HTTPS_PROXY="https://proxy.company.com:8443"

# Development and debugging (comment out in production)
# export CLAUDE_DEBUG=1
# export CLAUDE_LOG_LEVEL="INFO"

# === End Claude CLI Configuration ===
EOF

# Reload configuration
source ~/.zshrc

# Verify setup
claude auth status
claude config show

```

פקודת `claude config show` תציג את כל ההגדרות הפעילות, כולל אילו מהן נטענו ממשתני סביבה ואילו מקבצי קונפיגורציה.

2.4.4 ניהול מפתחות מרובים

במצבים בהם עובדים עם מספר חשבונות Anthropic (למשל, חשבון אישי וחשבון ארגוני), ניתן להשתמש בסקריפט עזר למעבר בין הגדרות:

```

# Create script: ~/bin/claude-switch-profile.sh
#!/bin/bash

```

```
PROFILE=$1
```

```
case $PROFILE in
  personal)
    export ANTHROPIC_API_KEY="sk-ant-personal-key"
    echo "Switched to personal profile"
    ;;
  work)
    export ANTHROPIC_API_KEY="sk-ant-work-key"
    export ANTHROPIC_BASE_URL="https://api.company-proxy.com"
    echo "Switched to work profile"
    ;;
  *)
    echo "Usage: claude-switch-profile.sh [personal | work]"
    exit 1
    ;;
esac
```

```
# Usage:
# source ~/bin/claude-switch-profile.sh personal
# source ~/bin/claude-switch-profile.sh work
```

שימו לב שיש להריץ את הסקריפט עם `source` כדי שמשתני הסביבה ייטענו ב-shell הנוכחי.

2.5 סיכום

בפרק זה התבססנו את היסודות הטכניים הנדרשים לעבודה עם Claude CLI. הבנו את היררכיית הקונפיגורציה התלת-רמתית, התקנו את הכלי בסביבה וירטואלית מבודדת, והגדרנו אימות מאובטח באמצעות משתני סביבה. עקרונות אלו מהווים את אבני היסוד לעבודה מקצועית עם כלי CLI מודרניים.

בפרק הבא נעמיק בהגדרות ברמת הפרויקט, נבחן את מבנה קובץ `CLAUDE.md`, ונלמד כיצד להתאים את התנהגות הכלי לצרכים ספציפיים של כל פרויקט.

2 הגדרת פרויקט ואבני בניין: מבנה התיקיות הבסיסי

בפרק הקודם הכרנו את העקרונות היסודיים של זרימת העבודה ה-Agentic, את סביבת Claude CLI כפלטפורמת פיתוח מקצועית, ואת הסוכן המתפתח שילווה אותנו – Yoram-Agent-Knowledge. כעת, הגיע הזמן לעבור מהתיאוריה אל המעשה. בפרק זה נניח את היסודות המעשיים הראשוניים: נבנה את מבנה התיקיות הפורמלי שעליו יעמוד הפרויקט כולו, נבין מדוע הקפדה על מבנה זה חיונית, ונגדיר את קובץ הקונפיגורציה הראשי שיקבע את ברירות המחדל לכלל הסוכנים בפרויקט שלנו.

עבור מהנדס התוכנה, זהו שלב מוכר: כל פרויקט מקצועי דורש מבנה תיקיות ברור ועקבי. עבור החוקר האקדמי, זוהי שכבת הארגון הנדרשת כדי לשמור על שחזוריות ותיעוד מדויק. ועבור הסטודנט, זוהי ההזדמנות ללמוד כיצד לבנות פרויקט בצורה נכונה מההתחלה, ולהימנע ממלכודות נפוצות של ארגון אד-הוק.

2.1 הסבר עקרוני: המבנה הקאנוני

כאשר אנו עובדים עם Claude CLI בסביבה Agentic, אנו נכנסים לעולם של **קונפיגורציה כקוד** (Configuration as Code). בניגוד לממשק גרפי שבו כל הגדרה נשמרת במסד נתונים פנימי וסגור, ב-CLI כל היכולות, ההרשאות והזהויות של הסוכן מוגדרות בקבצים מובנים – קבצי YAML, קבצי Markdown, סקריפטים של Python ומאגרי ידע טקסטואליים. זוהי גישה **שקופה, מתועדת וניתנת לבקרת גרסאות**, המאפשרת לנו לעקוב אחר כל שינוי, לשתף את הסוכן עם הצוות ולשכפל אותו לסביבות שונות.

אולם, כדי ש-Claude CLI יוכל לטעון את הקונפיגורציות הללו באופן אוטומטי, עלינו לארגן אותן במבנה תיקיות **קאנוני** – מבנה סטנדרטי שה-CLI מכיר ומחפש. מבנה זה אינו שרירותי: הוא משקף את הארכיטקטורה הפנימית של המערכת, ומבטיח שכל רכיב – סוכן, Skill, פקודה או מאגר ידע – יימצא במקום הצפוי.

מדוע זה חשוב? תארו לעצמכם פרויקט שבו קבצי הקונפיגורציה מפוזרים בתיקיות שונות ללא מבנה ברור. כל פעם שתמצאו להריץ סוכן, תצטרכו להפנות במפורש לכל קובץ, לזכור נתיבים, ולנהל באופן ידני את הקשרים ביניהם. זהו מתכון לבלבול, לשגיאות ולחוסר יכולת לשתף את הפרויקט עם אחרים. לעומת זאת, בעקבות המבנה הקאנוני, אנו מאפשרים ל-CLI לטעון הכל אוטומטית, בצורה היררכית ועקבית.

המבנה הקאנוני של Claude CLI מורכב מתיקיות קונפיגורציה מרכזיות `./claude/`. המכילה ארבע תיקיות ליבה, כל אחת בעלת תפקיד מוגדר:

1. `./claude/agents/` – תיקייה זו מכילה את קבצי ה-YAML או ה-Markdown המגדירים סוכנים ספציפיים. כל קובץ מייצג סוכן אחד, ומכיל את הזהות שלו (name, description), את המודל שבו הוא משתמש (model), את רשימת הכלים המותרים לו (tools), ואת הנחיות המערכת שלו (system_prompt). זוהי הלב הפועם של הפרויקט – כל סוכן שנרצה להריץ חייב להיות מוגדר כאן.

2. `./claude/skills/` – תיקייה זו מכילה תת-תיקיות, כאשר כל תת-תיקיה מייצגת Skill אחד – יכולת מותאמת אישית שניתן להוסיף לסוכן. בתוך כל תיקיית Skill נמצא

קובץ SKILL.md המגדיר את היכולת, את סכימת הקלט שלה (input_schema), ואת ההוראות לביצועה. בנוסף, Skill יכול לכלול סקריפטים של Python או קוד אחר. זוהי השכבה המאפשרת לנו להרחיב את יכולות הסוכן מעבר לכלים המובנים.

3. `claude/commands/` - תיקייה זו מכילה קבצי Slash Commands - פקודות מקוצרות שניתן להפעיל במהלך השיחה עם הסוכן. כל קובץ מייצג פקודה אחת, ומכיל את ההנחיות שיורחבו כאשר המשתמש יקליד את שם הפקודה. זוהי תוספת נוחות שמאפשרת זרימת עבודה יעילה יותר.

4. `knowledge/` - תיקייה זו מכילה קבצי טקסט או תיעוד שימשו את מנגנון ה-RAG (Retrieval-Augmented Generation) - הזיכרון הווקטורי של הסוכן. כאשר נגדיר RAG בפרקים מתקדמים יותר, ה-CLI יאנדקס באופן אוטומטי את כל הקבצים בתיקייה זו ויאפשר לסוכן לשלוף מידע מתוכם בזמן אמת. זוהי השכבה המאפשרת לסוכן לגשת לידע פנימי שחורג מחלון ההקשר של המודל.

שימו לב: המבנה הזה אינו חובה טכנית לכל פרויקט CLI, אך הוא **מומלץ בחום** ונחשב ל-best practice מקצועי. הקפדה עליו מבטיחה שכל מי שיקבל את הפרויקט - בין אם זה עמית בצוות, חוקר אחר או אפילו אתם בעצמכם בעוד חצי שנה - יבין מיד את המבנה ויוכל לעבוד איתו ללא בלבול.

2.2 ההגדרה המעשית: יצירת מבנה התיקיות

כעת נעבור לשלב המעשי: נבנה את מבנה התיקיות עבור הפרויקט הדוגמה שלנו, Knowledge-Agent-Yoram. שם זה נבחר בקפידה: הוא משקף את תפקידו של הסוכן - ניהול ידע מתקדם - ומכיל את השם "יורם", שלפי הנחיות הפרויקט מחייב **טון מקצועי ורשמי** בכל התיעוד והתקשורת. זהו סוכן אנליסט בכיר, לא עוזר אישי קליל, ועלינו לשמור על אופי זה בכל שכבות הפרויקט.

הערה חשובה: המבנה הקאנוני של Claude CLI דורש שכל קבצי הקונפיגורציה יהיו תחת תיקייה מיוחדת בשם `claude/`. בשורש הפרויקט. תיקייה זו (המתחילה בנקודה) היא תיקייה מוסתרת במערכות Unix/Linux, ומכילה את כל הגדרות הסוכנים, ה-Skills והפקודות.

שלב ראשון: יצירת תיקיית הפרויקט הראשית

נתחיל ביצירת תיקייה ראשית שתשמש כבסיס עבודה לכל הפרויקט. תיקייה זו תכיל את כל הרכיבים - הסוכנים, ה-Skills, הפקודות ומאגרי הידע. בחירת מיקום התיקייה תלויה בסביבת העבודה שלכם: אם אתם עובדים בסביבת Linux או macOS, ייתכן שתמצאו את הפרויקט תחת `~/projects/` או נתיב דומה. בסביבת Windows, ייתכן שתעדיפו תיקייה כגון `C:\Projects\`. בכל מקרה, העיקרון זהה. להלן דוגמת פקודת Bash ליצירת התיקייה הראשית:

```
# טקדורפה תיקיית תרצוי 1.
mkdir Knowledge-Agent-Yoram
```

```
# תישארה הייקיתה דותל רבעמ 2.
cd Knowledge-Agent-Yoram
```

לאחר ביצוע פקודות אלו, אנו נמצאים בתוך תיקיית הפרויקט הריקה. זהו נקודת המוצא שלנו.

שלב שני: יצירת התיקיות הקאנוניות

כעת ניצור את תיקיית הקונפיגורציה המרכזית `.claude/` ותחתיה את שלוש תיקיות הליבה. בנוסף, ניצור את תיקיית `knowledge/` בשורש הפרויקט. כל תיקייה תהיה ריקה בשלב זה, אך נוכחותה תאפשר ל-CLI לזהות את הפרויקט כמובנה כראוי.

```
# תיזכרמה היצורוגיפנוקה תייקית תריצי 3.
mkdir .claude
```

```
# תחת תוינונאקה תויקיתה תריצי 4.
mkdir .claude/agents
mkdir .claude/skills
mkdir .claude/commands
```

```
# טקיורפה תמרב) עדיה תייקית תריצי 5.
mkdir knowledge
```

שלב שלישי: אימות המבנה

לאחר יצירת התיקיות, מומלץ לאמת שהמבנה נוצר כראוי. ניתן לעשות זאת באמצעות פקודת `tree` (או `ls -R` אם `tree` לא זמינה):

```
# רצונש תויקיתה הנבמ תקידב 6.
tree -L 2
```

```
# היופצה האצותה:
# Knowledge-Agent-Yoram/
# |
# |   ├── .claude/
# |   |   ├── agents/
# |   |   ├── commands/
# |   |   └── skills/
# |   └── knowledge/
```

אם אתם רואים את המבנה הזה, מעולה – התשתית הבסיסית מוכנה. שימו לב שתיקיות הקונפיגורציה (`agents`, `skills`, `commands`) נמצאות תחת `.claude/` ולא בשורש הפרויקט. זהו המבנה הסטנדרטי של Claude CLI.

סיכום ביניים: מה יש לנו כרגע?

בשלב זה, יש לנו תיקיית פרויקט עם מבנה היררכי ברור: תיקיית /claude. מרכזת את כל הקונפיגורציות, ותיקיית /knowledge מכילה את מאגרי הידע. זהו שלד הפרויקט – מבנה התשתית שעליו נבנה את כל השכבות הבאות. הטבלה הבאה (טבלה 1) מסכמת את מבנה התיקיות הקאנוני:

טבלה 1: מבנה תיקיות הפרויקט Knowledge-Agent-Yoram

תפקיד	Location	רכיב
בסיס העבודה	Knowledge-Agent-Yoram/	תיקיית פרויקט
תיקיית הראשית Claude CLI	.claude/	קונפיגורציה מרכזית
קבצי המגדירים YAML/Mark-down סוכנים	.claude/agents/	הגדרות סוכנים
תיקיות עבור כל Skill מותאם אישית	.claude/skills/	יכולות מורחבות
קבצים עבור Slash Commands	.claude/commands/	פקודות סלאש
קבצי טקסט לשימוש ב-RAG	knowledge/	זיכרון וידע

שימו לב לנקודה חשובה: כל תיקיות הקונפיגורציה (agents, skills, commands) נמצאות תחת /claude, בעוד תיקיית /knowledge נמצאת בשורש הפרויקט. זהו המבנה הסטנדרטי של Claude CLI, והקפדה עליו מבטיחה שה-CLI יזהה את הקונפיגורציות אוטומטית.

2.3 הדגמה: קובץ הקונפיגורציה הראשי

עד כה יצרנו את מבנה התיקיות הבסיסי, אך הפרויקט עדיין חסר הגדרות כלליות. כעת נוסיף רכיב אופציונלי אך מומלץ ביותר: קובץ `project.yaml` – קובץ קונפיגורציה ראשי ברמת הפרויקט, שמטרתו לקבוע **ברירות מחזל** לכלל הסוכנים בפרויקט זה.

מדוע זה שימושי? תארו לעצמכם פרויקט המכיל מספר סוכנים שונים – סוכן לניתוח מסמכים, סוכן לסיכום דוחות, סוכן למחקר. אם כל סוכן צריך להגדיר מחדש את אותו מודל (Opus, Sonnet) ואת אותם כלים בסיסיים (read, webfetch), נצטרך לשכפל את ההגדרות הללו בכל קובץ YAML של סוכן. זה לא רק מייגע – זה גם מגדיל את הסיכון לטעויות ולחוסר עקביות.

קובץ ה-`config.json` (או `project.yaml`) פותר בעיה זו באמצעות **ירושה**. כל סוכן שנגדיר בתיקיית /claude/agents/ יירש באופן אוטומטי את ההגדרות מקובץ זה, אלא אם נדרוס אותן

במפורש בקובץ הסוכן עצמו. זוהי גישה **היררכית** המאפשרת גמישות ועקביות במקביל.

יצירת קובץ config.json

ניצור קובץ בשם config.json בתיקיית .claude/ (היינו, תחת .claude/config.json). קובץ זה יהיה בפורמט JSON – פורמט מובנה ופשוט לקריאה, המשמש באופן נרחב ב-Claude CLI להגדרות פרויקט. להלן תוכן דוגמה לקובץ:

```
// .claude/config.json
{
  "model": "sonnet",
  "global_tools": ["read", "webfetch"]
}
```

הסבר השדות:

model – שדה זה קובע את מודל ברירת המחדל לכל הסוכנים בפרויקט. בדוגמה שלנו, בחרנו במודל "sonnet" – מודל מאוזן בין יכולות ומהירות, המתאים למשימות יומיומיות. בפרקים מתקדמים יותר, כאשר נגדיר את הסוכן expert-analyst עצמו, נדרוש הגדרה זו ונציין במפורש שהוא צריך להשתמש במודל "opus" – המודל החזק ביותר, המתאים לניתוח מורכב.

global_tools – שדה זה מגדיר רשימה של כלים בסיסיים שיהיו זמינים לכל הסוכנים בפרויקט. בדוגמה שלנו, הגדרנו שני כלים חיוניים:

read – יכולת קריאת קבצים מהפרויקט או מהמערכת.

webfetch – יכולת חיפוש מידע באינטרנט ושליפת תוכן מכתובות URL.

שני כלים אלו נחשבים בסיסיים כמעט לכל סוכן ניתוח או מחקר, ולכן הגיוני להגדיר אותם כלליים ברמת הפרויקט.

איך זה עובד בפועל?

כאשר נריץ סוכן כלשהו מתוך תיקיית .claude/agents/, ה-CLI יבצע את השלבים הבאים:

1. **טעינת הגדרות ברמת המערכת** – אם קיימות הגדרות גלובליות במערכת (למשל, מודל ברירת מחדל כללי).

2. **טעינת הגדרות ברמת הפרויקט** – ה-CLI יחפש קובץ .claude/config.json בתיקיית הפרויקט, ויטעון את model ו-global_tools.

3. **טעינת הגדרות ברמת הסוכן** – ה-CLI יקרא את קובץ ה-YAML של הסוכן עצמו (למשל, .claude/agents/expert-analyst.yaml), ויבדוק אם הוגדרו שם שדות model או tools.

4. **מיזוג וקביעת עדיפות** – אם הסוכן עצמו הגדיר מודל ספציפי, הוא ידרוס את ה-model מקובץ הקונפיגורציה. אם הסוכן הגדיר רשימת tools, היא תתווסף ל-global_tools (או תדרוס אותה, תלוי בהגדרות המערכת).

זוהי היררכיה ברורה: **מערכת** → **פרויקט** → **סוכן**, כאשר כל רמה יכולה לדרוס את קודמתה. גישה זו מאפשרת גמישות מרבית: אנו יכולים להגדיר ברירות מחדל סבירות ברמת הפרויקט, ולהתאים אותן למקרים ספציפיים ברמת הסוכן.

דוגמה מעשית להמחשה:

נניח שיצרנו שני סוכנים בפרויקט:

1. **quick-summary** – סוכן לסיכומים מהירים, שלא מגדיר מודל משלו. הוא יירש את "sonnet" מ-`./claude/config.json`, ויקבל את הכלים `read` ו-`webfetch`.

2. **expert-analyst** – הסוכן המתקדם שלנו, שבקובץ ה-YAML שלו (`./claude/agents/expert-analyst.yaml`) מוגדר במפורש `model: "opus"`. הוא ידרוס את ברירת המחדל, ויקבל את Opus במקום Sonnet. בנוסף, הוא יכול להוסיף כלים נוספים (כגון `grep`, `structured-report`) על גבי הכלים הגלובליים.

זוהי הדרך המקצועית לנהל פרויקט רב-סוכנים: הגדרות משותפות ברמת הפרויקט, התאמות ספציפיות ברמת הסוכן.

הערת אבטחה חשובה:

קובץ `./claude/config.json` אינו המקום הנכון לשמור מפתחות API או סודות אחרים. מפתח ה-API של Anthropic צריך להישמר במשתנה סביבה (`ANTHROPIC_API_KEY`), כפי שהוסבר בפרק הקודם. קובץ זה מיועד **רק להגדרות לוגיות** – מודלים, כלים, מדיניות – ולא לנתונים רגישים.

2.4 סיכום ומבט קדימה

בפרק זה ביצענו את הצעדים המעשיים הראשונים לקראת בניית הסוכן שלנו. יצרנו את מבנה התיקיות הקאנוני – תיקיית `./claude/` המכילה שלוש תיקיות ליבה (`agents/`, `skills/`, `commands/`), ותיקית `knowledge/` בשורש הפרויקט. זהו השלד הסטנדרטי של פרויקט Knowledge-Agent-Yoram. הבנו מדוע הקפדה על מבנה זה חיונית: היא מאפשרת ל-CLI לטעון את הקונפיגורציות אוטומטית, מבטיחה שחזריות ושקיפות, ומקלה על שיתוף הפרויקט ועל עבודה צוותית.

בנוסף, יצרנו קובץ `./claude/config.json` – קובץ קונפיגורציה ראשי שקובע ברירות מחדל לכלל הסוכנים בפרויקט. הגדרנו את מודל ברירת המחדל (`sonnet`) ואת הכלים הגלובליים (`read`, `webfetch`), תוך הבנה שכל סוכן יוכל לדרוס הגדרות אלו לפי צורכיו הספציפיים.

עבור מהנדס התוכנה, שלבים אלו צריכים להיות מוכרים ואינטואיטיביים – זהו עיקרון בסיסי בארגון פרויקטים. עבור החוקר האקדמי, זוהי השכבה הארגונית הנדרשת כדי לשמור על תיעוד מדויק ושחזריות ניסיונית. ועבור הסטודנט, זוהי ההזדמנות ללמוד כיצד

לבנות פרויקט בצורה נכונה מההתחלה – מבנה ברור, קונפיגורציה שקופה, ותשתית מוכנה להתרחבות.

בפרק הבא, **נבנה את הלב הפועם של הפרויקט** – קובץ ה-YAML של הסוכן expert-analyst עצמו. נגדיר את הזהות שלו (name, description), נבחר את המודל המתאים (Opus), ונכתוב את הנחיות המערכת (System Prompt) שיעצבו את אישיותו ואת צורת התנהגותו. זהו השלב שבו הסוכן מקבל חיים – מעבר ממבנה תיקיות ריק לישות אינטליגנטית בעלת מטרה ויכולות.

המבנה מוכן. הבה נמשיך לבנות.

3 השלד: בניית קובץ ה-Agent YAML

בפרק זה נעבור מהתכנון לביצוע ממשי. לאחר שהכנו את מבנה התיקיות והקמנו את סביבת העבודה הבסיסית, הגיע הזמן להגדיר את **הזהות המרכזית** של הסוכן שלנו. קובץ ה-Agent YAML הוא אבן היסוד שעליה נבנה כל שאר היכולות – הוא מגדיר את השם, המטרה, המודל ואת האישיות הבסיסית של הסוכן.

3.1 הסבר עקרוני: קובץ הקונפיגורציה (Agent Manifest)

קובץ ה-Agent YAML של הסוכן (Agent Manifest) הוא הלב הפועם של כל סוכן ב-Claude CLI [7]. קובץ זה מגדיר את הזהות, המטרה והיכולות המותרות לסוכן הספציפי. בהתאם למידע שסופק, קובץ זה יכול להיות קובץ YAML טהור, או Frontmatter בקובץ Markdown (כגון agent-name.md). במדריך זה, נשתמש בפורמט YAML [8] פשוט כדי להדגים את הליבה הקונפיגורטיבית בצורה הברורה ביותר.

3.1.1 הגדרות בסיסיות חובה

כל סוכן חייב להכיל ארבעה רכיבים מרכזיים:

1. **name** – מזהה ייחודי שבו נשתמש כדי לקרוא לסוכן (חובה).
 2. **description** – תיאור קצר, המגדיר את מטרת הסוכן ואת תפקידו (חובה).
 3. **model** – ציון מודל ה-Claude הספציפי לשימוש הסוכן (דורס את ה-default_model ב-project.yaml).
 4. **tools** – רשימה זו תכיל את הכלים וה-Skills המותרים לסוכן.
- בנוסף לרכיבים אלה, נגדיר גם את ה-system_prompt – ההנחיה ברמת המערכת המגדירה את אישיות הסוכן ואת כללי ההתנהגות המחייבים אותו.

3.1.2 מבנה קובץ ה-Agent YAML

המבנה הבסיסי של קובץ Agent YAML עבור סוכן נראה כך:

Listing 1: LMay tnegA של קובץ מבנה בסיסי

```
---
name: "agent-name"
description: "Brief description of the agent's purpose"
model: "opus" # or "sonnet" or "haiku"
tools: [] # List of allowed tools and skills
system_prompt: |
    System-level instructions that define the agent's
    personality and behavior guidelines.
```

שימו לב ל-`system_prompt` שמשמש בתחביר | המאפשר טקסט רב-שורתי ב-YAML, דבר חיוני להגדרת הנחיות מפורטות.

3.2 ההגדרה המעשית: יצירת ה-YAML לסוכן ניהול הידע

ניצור כעת את קובץ הקונפיגורציה עבור הסוכן שלנו, `Yoram-Agent-Knowledge`. כזכור מפרק 2, מטרת סוכן זה היא לשמש כאנליסט ידע מומחה – מטרתו לשלוף, לסכם ולנתח מסמכים פנימיים וחיצוניים, תוך שמירה על דיוק מקצועי מוחלט. שם הסוכן "`maroY`" נבחר במכוון כדי לשקף טון רשמי ומקצועי, בהתאם לדרישות הפרויקט. כל התקשורת של סוכן זה תהיה מקצועית, שיטתית ורשמית.

3.2.1 שלב א': יצירת קובץ ה-YAML

נבצע את הפעולות הבאות ליצירת קובץ הסוכן:

Listing 2: יצירת קובץ `lmay.tsylana-trepxe`

```
# Navigate to project directory
cd Knowledge-Agent-Yoram/

# Create YAML file in agents/ directory
touch agents/expert-analyst.yaml
```

3.2.2 שלב ב': הזנת תוכן השלד

כעת נזין את הקונפיגורציה הבסיסית לקובץ `agents/expert-analyst.yaml`:

Listing 3: קובץ `lmay.tsylana-trepxe` המלא

```
# Knowledge-Agent-Yoram/agents/expert-analyst.yaml
---
name: "expert-analyst"
description: "חותינ, תמדקתמ, סינותנ, תפילשל, דעוימה, החמומ, נכוס"
קויד, לע, רומשל, ביוחמ, הז, נכוס. יקמ, עדימ, סוכיס, סיינכט, סיכמסמ,
" . טלחומ, יעוצקמ,
model: "opus" # תובכרומ חותינ תומישמל רתויב סדקתמה לדומב רחבנ
tools: [] # 4 קרפב סילכ פיסונו, עגרכ קיר ריאשנ
system_prompt: |
    קמונמו תמואמ תויהל בייח טלפ לכ. קיודמו ריכב טסילנא התא
    .פלשנ וא קפוסש עדימה לע ססבתהב

    תיעוצקמ, תיתטיש תויהל תבייח דלש השיגה. תויצלוקפסמ ענמיה
    .תימשרו
```

3.2.3 דגשים על ההגדרות

נבחן כעת כל שדה בקובץ ונבין את משמעותו:

1. `name: "expert-analyst"` זהו השם שאיתו נפעיל את הסוכן מהשורה הפקודה:

```
claude agent run expert-analyst --task "..."
```

השם חייב להיות ייחודי בתוך תיקיית `agents/` של הפרויקט.

2. `model: "opus"` הגדרה זו דורסת את ברירת המחדל ("`sonnet`") שהוגדרה ב-`project.yaml`, ומבטיחה שימוש במודל הארגוני החזק ביותר (Opus) לניתוח מורכב. מודל זה מתאים במיוחד למשימות הדורשות חשיבה עמוקה, ניתוח רב-שלבי ודיוק גבוה.

3. `description` התיאור מסביר בקצרה את מטרת הסוכן. הוא משמש לתיעוד פנימי ועוזר למשתמשים אחרים (או למערכת) להבין מתי להשתמש בסוכן זה. שימו לב שהתיאור כתוב בעברית, מה שמדגיש את המקצועיות והנגישות לקהל דוברי העברית.

4. `system_prompt` זוהי ההנחיה ברמת המערכת (System Prompt), המגדירה את אישיות הסוכן ואת כללי ההתנהגות המחייבים אותו. ההנחיה הזו מוזנת למודל בכל שיחה כחלק מההקשר הבסיסי, ומשפיעה ישירות על סגנון התשובות, הטון והגישה המקצועית.

שימו לב לשימוש בטון מקצועי ורשמי, בהתאם לבקשה המקורית אודות השימוש בשם 'maroY' עבור טקסט רשמי. ההנחיה מדגישה:

- דיוק מוחלט – כל תשובה חייבת להיות מאומתת
- הימנעות מספקולציות – אין לנחש או להמציא מידע
- גישה שיטתית ומקצועית – תהליך עבודה מסודר ורשמי

5. `tools: []` בשלב זה, רשימת הכלים ריקה. בפרק 4, נוסף כלים מובנים כגון `websearch`, `read` ו-`grep`. בפרקים מתקדמים יותר, נוסף גם Skills מותאמים אישית.

3.3 אימות הקובץ

לאחר יצירת הקובץ, חשוב לוודא שתחביר ה-YAML תקין. ניתן לעשות זאת באמצעות סקריפט Python פשוט שיאמת את המבנה ויבדוק שכל השדות החובה קיימים.

3.3.1 שימוש ב-Python לאימות YAML

ניצור סקריפט קצר שיקרא את קובץ ה-YAML, יאמת את תקינותו ויצג את התוכן בצורה מובנית:

Listing 4: סקריפט אימות קובץ LMay

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Validate Agent YAML Configuration
Checks syntax and required fields
"""

import yaml
import sys

def validate_agent_yaml(filepath):
    """Validate agent YAML file structure"""
    try:
        with open(filepath, 'r', encoding='utf-8') as f:
            config = yaml.safe_load(f)

        # Check required fields
        required_fields = ['name', 'description', 'model']
        missing = [f for f in required_fields if f not in config]

        if missing:
            print(f"Error: Missing required fields: {missing}")
            return False

        # Display configuration
        print("Agent Configuration Valid:")
        print(f"  Name: {config['name']}")
        print(f"  Model: {config['model']}")
        print(f"  Description: {config['description'][:50]}...")
        print(f"  Tools: {len(config.get('tools', []))}")

        return True

    except yaml.YAMLError as e:
```

```

        print(f"YAML_Syntax_Error: {e}")
        return False
    except FileNotFoundError:
        print(f"File_not_found: {filepath}")
        return False

if __name__ == '__main__':
    filepath = 'agents/expert-analyst.yaml'
    if len(sys.argv) > 1:
        filepath = sys.argv[1]

    valid = validate_agent_yaml(filepath)
    sys.exit(0 if valid else 1)

```

הרצת הסקריפט:

```
python src/validate_agent_yaml.py agents/expert-analyst.yaml
```

פלט צפוי:

Agent Configuration Valid:

Name: expert-analyst

Model: opus

Description: ...תמדקתם סינותנ תפילשל דעוימה החמום וכוס

Tools: 0

3.4 יצירת מחולל YAML אוטומטי

כדי להקל על יצירת סוכנים נוספים בעתיד, ניצור סקריפט Python שמנחה את המשתמש בתהליך אינטראקטיבי ומייצר קובץ YAML תקין אוטומטית. סקריפט זה יבקש מהמשתמש להזין את שם הסוכן, התיאור, המודל המועדף וה-system prompt, וייצור קובץ מוכן לשימוש.

הסקריפט מופיע בקוד 5 ומממש את הפונקציונליות הבאה:

- קבלת קלט אינטראקטיבי מהמשתמש

- אימות תקינות השדות

- יצירת קובץ YAML תקני

- שמירה בתיקייה המתאימה

הערה חשובה: הסקריפט מוגבל ל-150 שורות, בהתאם להנחיות הפרויקט הכלליות.

Listing 5: מחולל קובצי LMAy tneGA אוטומטי

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Module: create_agent_yaml.py
Chapter: 03
Purpose: Interactive agent YAML file generator

Part of: AI Agent Book - Building Custom AI Agents
Max Lines: 150 (MANDATORY)
"""

import os
import yaml
from pathlib import Path

def get_user_input(prompt, default=None):
    """Get input from user with optional default"""
    if default:
        prompt = f"{prompt} [{default}]:"
    else:
        prompt = f"{prompt}:"

    value = input(prompt).strip()
    return value if value else default

def select_model():
    """Interactively select Claude model"""
    print("\nSelect Claude model:")
    print("1. opus - Most powerful (expensive)")
    print("2. sonnet - Balanced (recommended)")
    print("3. haiku - Fast (economical)")

    choice = get_user_input("Enter choice", "2")

    models = {"1": "opus", "2": "sonnet", "3": "haiku"}
    return models.get(choice, "sonnet")
```



```

def create_agent_yaml(
    name,
    description,
    model,
    system_prompt,
    tools=None,
    output_dir="agents"
):
    """
    Create agent YAML file

    Args:
        name: Agent unique identifier
        description: Agent purpose description
        model: Claude model (opus/sonnet/haiku)
        system_prompt: System-level instructions
        tools: List of allowed tools (default: [])
        output_dir: Output directory (default: 'agents')

    Returns:
        Path to created file
    """
    if tools is None:
        tools = []

    # Prepare configuration dictionary
    config = {
        'name': name,
        'description': description,
        'model': model,
        'tools': tools,
        'system_prompt': system_prompt
    }

    # Create output directory if needed
    Path(output_dir).mkdir(parents=True, exist_ok=True)

```

```

# Generate filename
filename = f"{name}.yaml"
filepath = Path(output_dir) / filename

# Write YAML file
with open(filepath, 'w', encoding='utf-8') as f:
    f.write("#_Agent_Configuration\n")
    f.write(f"#_Generated_by_create_agent_yaml.py\n")
    f.write("---\n")
    yaml.dump(
        config,
        f,
        default_flow_style=False,
        allow_unicode=True,
        sort_keys=False
    )

return filepath


def interactive_mode():
    """Interactive_agent_creation_wizard"""
    print("=" * 60)
    print("Agent_YAML_Generator_-_Interactive_Mode")
    print("=" * 60)

    # Get agent details
    name = get_user_input("Agent_name_(e.g.,_expert-analyst)")
    if not name:
        print("Error:_Name_is_required")
        return None

    description = get_user_input(
        "Agent_description",
        "A_specialized_AI_agent_for_specific_tasks"
    )

    model = select_model()

```

```

print("\nEnter system prompt (multi-line).")
print("Finish with empty line:")
lines = []
while True:
    line = input()
    if not line:
        break
    lines.append(line)

system_prompt = "\n".join(lines) if lines else \
    "You are a helpful and professional assistant."

# Create YAML file
filepath = create_agent_yaml(
    name=name,
    description=description,
    model=model,
    system_prompt=system_prompt
)

print(f"\nAgent YAML created: {filepath}")
return filepath

def main():
    """Main entry point"""
    import sys

    if len(sys.argv) > 1 and sys.argv[1] == '--interactive':
        interactive_mode()
    else:
        print("Usage:")
        print("python create_agent_yaml.py --interactive")
        print("\nOr use as module:")
        print("from create_agent_yaml import create_agent_yaml")

if __name__ == '__main__':
    main()

```

3.4.1 שימוש במחולל

להלן דוגמה להפעלת המחולל במצב אינטראקטיבי:

Listing 6: הרצת מחולל LMAY במצב אינטראקטיבי

```
python src/create_agent_yaml.py --interactive
```

המערכת תנחה אתכם בשלבים הבאים:

1. הזנת שם הסוכן
 2. הזנת תיאור המטרה
 3. בחירת מודל (opus/sonnet/haiku)
 4. הזנת ה-system prompt (רב-שורתי)
- בסיום, הקובץ ייוצר אוטומטית בתיקייה agents/ ויהיה מוכן לשימוש.

3.5 סיכום הפרק

בפרק זה למדנו את היסודות של הגדרת סוכן AI באמצעות קובץ YAML:

1. הכרנו את מבנה קובץ ה-Agent Manifest והשדות החובה שלו
2. יצרנו את קובץ הסוכן expert-analyst.yaml עבור egdelwonK-tnegA-maroy
3. הגדרנו זהות מקצועית ורשמית באמצעות system_prompt מותאם
4. למדנו לאמת תקינות קובצי YAML באמצעות Python
5. יצרנו מחולל אוטומטי ליצירת סוכנים נוספים בעתיד

בפרק הבא (4), נרחיב את יכולות הסוכן שלנו על ידי הוספת **כלים מובנים** (Built-in Tools) כגון read, websearch ו-grep, שיאפשרו לו לגשת למידע, לקרוא קבצים ולחפש באינטרנט.

זכרו: קובץ ה-YAML הוא הבסיס לכל סוכן. השקעת זמן בהגדרה נכונה של הזהות, המודל וה-system prompt תשפיע ישירות על איכות התוצאות ועל מקצועיות הסוכן שלכם.

חלק II

חלק שני: כלים ושילוב

4 כלים מובנים

לאחר שהגדרנו את זהותו הבסיסית של הסוכן באמצעות קובץ ה-YAML, הגיע הזמן להעניק לו יכולות אמיתיות. בפרק זה נבחן את הכלים המובנים (Built-in Tools) שמספק Claude CLI, נבין את עקרון ההרשאות והאימות, ונוסיף כלים חיוניים לסוכן expert-analyst שלנו. התהליך של הוספת כלים לסוכן דומה להעניקת חושים לאורגניזם: כל כלי מרחיב את תחום התפיסה והפעולה של הסוכן בעולם הדיגיטלי.

4.1 הסבר עקרוני: אימוץ כלים פנימיים

Claude CLI מגיע עם ארגז כלים עשיר ומגוון, כאשר כל כלי מספק יכולת ספציפית. הכלים המובנים עוברים בדיקות קפדניות מבחינת אבטחה ויעילות, ומהווים את אבני הבניין הבסיסיות לכל סוכן. בניגוד לכלים מותאמים אישית שנפתחו בפרקים הבאים, הכלים המובנים מוכנים לשימוש מיידי ואינם דורשים הגדרה נוספת מעבר להכללתם ברשימת הכלים של הסוכן.

4.1.1 פילוסופיית הכלים: יכולות מול אחריות

כל כלי שאנו מוסיפים לסוכן מהווה הרחבה משמעותית של יכולותיו, אך גם של אחריותנו כמפתחים. עקרון Principle of Least Privilege - עקרון ההרשאה המינימלית - מחייב אותנו להעניק לסוכן רק את הכלים שהוא באמת זקוק להם לביצוע משימותיו. זהו איזון עדין: מצד אחד, אנו רוצים שהסוכן יהיה עצמאי ויעיל; מצד שני, אנו חייבים למנוע מצב שבו סוכן עם גישה רחבה מדי עלול לגרום נזק בשוגג או במכוון. נניח לדוגמה שאנו מפתחים סוכן לניתוח לוגים (log analysis agent). סוכן כזה זקוק לכלי read כדי לקרוא קבצי לוג, אך אינו זקוק לכלי write או bash. הוספת כלים מיותרים עלולה ליצור משטח תקיפה רחב יותר במקרה של התנהגות לא צפויה של המודל.

4.1.2 קטגוריות הכלים המובנים

הכלים המובנים של Claude CLI מחולקים למספר קטגוריות פונקציונליות:

- כלי קריאת מידע (Information Retrieval):** כלים המאפשרים לסוכן לאסוף נתונים ממקורות שונים, כגון read (קריאת קבצים), websearch (חיפוש באינטרנט), ו-webfetch (טעינת תוכן מ-URL ספציפי).
- כלי חיפוש וסינון (Search and Filter):** כלים לאיתור דפוסים בקבצים ובמבני תיקיות, כגון grep (חיפוש טקסט) ו-glob (חיפוש קבצים לפי תבניות).
- כלי כתיבה ועריכה (Writing and Editing):** כלים המאפשרים שינוי קבצים, כגון write (יצירת/שכתוב קבץ) ו-edit (עריכה חכמה של קבץ קיים).
- כלי הרצה (Execution Tools):** כלים להרצת פקודות מערכת, כגון bash (הרצת פקודות shell). כלים אלו דורשים זהירות מיוחדת והם יידונו בפירוט בפרק ?? העוסק באבטחה.
- כלי ניהול משימות (Task Management):** כלים לארגון עבודה, כגון todowrite לניהול רשימות משימות.

4.1.3 שלושת הכלים החיוניים: read, websearch, grep

עבור סוכן מחקר אקדמי כמו expert-analyst, שלושה כלים מהווים את הליבה המינימלית הנדרשת לעבודה יעילה. נבחן כל אחד מהם לעומק.

כלי read: קריאת קבצים

הכלי read מאפשר לסוכן לקרוא תוכן של קבצים במערכת הקבצים המקומית. זהו הכלי הבסיסי ביותר והחיוני ביותר, שכן בלעדיו הסוכן "עיוור" למסמכים הקיימים בפרויקט.

יכולות: קריאת קבצי טקסט בפורמטים שונים (.txt, .md, .py, .json, .yaml), קריאת תמונות (.png, .jpg) באמצעות ראייה ממוחשבת, קריאת קבצי PDF ו-Jupyter notebooks (ipynb).

מקרי שימוש: ניתוח קוד מקור, סקירת תיעוד, עיבוד תוצאות מחקר, ניתוח לוגים.

הגבלות אבטחה: הכלי מוגבל לקריאת קבצים בלבד ולא יכול לשנות תוכן. הוא כבד את הרשאות המערכת - אם המשתמש המריץ את הסוכן אינו יכול לקרוא קובץ מסוים, גם הסוכן לא יוכל.

כלי websearch: חיפוש באינטרנט

הכלי websearch מעניק לסוכן גישה לידע עדכני מהאינטרנט. זהו כלי קריטי עבור סוכני מחקר, שכן המודלים השפתיים עצמם מוגבלים לידע עד תאריך גזירת הידע שלהם (knowledge cutoff).

יכולות: ביצוע חיפוש סמנטיים ברשת, איסוף תוצאות רלוונטיות, סינון לפי דומינים (allowed_domains, blocked_domains), הגבלת תוצאות לתאריכים ספציפיים.

מקרי שימוש: איסוף נתונים עדכניים למחקר, אימות עובדות, איתור מקורות אקדמיים, מעקב אחר טכנולוגיות חדשות.

שיקולים: החיפוש מבוסס על שירותי חיפוש חיצוניים ועלול להיות כפוף להגבלות אזוריות. בנוסף, יש להיות מודעים לכך שתוצאות החיפוש עשויות להכיל מידע לא מדויק - הסוכן חייב להפעיל שיקול דעת ביקורתי.

כלי grep: חיפוש תבניות בקבצים

הכלי grep (מבוסס על ripgrep המהיר במיוחד) מאפשר חיפוש ביטויים רגולריים בקבצים ובמבני תיקיות. זהו כלי עוצמתי לסריקת קוד מקור, איתור תבניות, וניתוח מבני.

יכולות: חיפוש ביטויים רגולריים מורכבים, סינון לפי סוג קובץ (type: "py" עבור קבצי Python), סינון לפי תבניות glob ("*.tsx"), מצבי פלט שונים (files_with_matches, content, count), הצגת הקשר (-A, -B, -C) עבור שורות לפני/אחרי/סביב התוצאה.

מקרי שימוש: איתור כל המקומות בהם משתמש מסוים מוזכר בקוד, ספירת שימושים בפונקציה, איתור קבצים המכילים תבנית שגיאה, ניתוח מבנה פרויקט.

התחביר: הכלי משתמש בתחביר ripgrep, לא grep רגיל. לדוגמה, סוגריים מסולסלים דורשים escaping: interface\{\} למציאת interface{} בקוד Go.

4.2 ההגדרה המעשית: הוספת הכלים לקובץ ה-YAML

לאחר שהבנו את התפקיד של כל כלי, נעבור לשלב המעשי: הוספתם לקובץ expert-analyst.yaml של הסוכן שלנו.

4.2.1 מבנה רשימת הכלים

רשימת הכלים מוגדרת תחת המפתח tools בקובץ ה-YAML, כרשימה פשוטה של מחרוזות. כל כלי מזוהה על-ידי שמו הייחודי.

```
# expert-analyst.yaml - Version with Built-in Tools

name: "expert-analyst"

description: |
    מיכמסמ חותינ ,תמדקתמ סינותנ תפילשל דעוימה החמומ נכוס
    קויד לע רומשל ביוחמ הז נכוס .פיקמ עדימ מוכיסו מיינכט
    טלחומ יעוצקמ

model: "claude-sonnet-4-5-20250929"

# Built-in tools for information retrieval and analysis
tools:
  - read          # Read local files
  - websearch     # Search the web for current information
  - grep          # Search patterns in files and directories

system_prompt: |
    יעוצקמ עדימ חותינב החמתמה קיודמו ריכב טסילנא התא

    ## תיעוצקמ השיגו תוהז
    מינש בר וויסינ לעב החמומ טסילנא - סרוי אוה יעוצקמה כמש
    ,כלש טלחומה קוידב עודי התא .עדימ דוביעו ימדקאה רקחמה מוחתב
    .יעוצקמו ילמרופ תרושקת ונגסבו ,סינותנ חותינל תיתטיש השיגב

    ## הדובע תונורקע
    1. חוטב דניא סא .עדימ איצמת וא רעשת לא :טלחומ קויד
       שרופמב ככ לע רהצה
    2. תיללכ הריקסמ לחה ,הנבומ נפואב עדימ חתנ :תיתטיש השיג
       מייפיצפסה מיטרפל דעו
    3. רוקמב הבוגמ תויהל בייח עדימ עטק לכ :תורוקמ דועית
       .נימאו ההוזמ
    4. תיעוצקמו תילמרופ תירבעב שמתשה :תילמרופ הפש
       .ליגר רובידמ ענמיה
```


מילכב שומיש

:מיינויח מילכ השולשב דיוצמ התא

- ****read****: מיינוקמ סיכמסמ תאירקל שמתשה , מינותנ יצבק , תוססובמ תונבות קפהו יתטיש נפואב מיצבק ארק . רקחמ תואצותו
- ****websearch****: טנרטניאהמ ינכדע עדימ פוסיאל שמתשה . תונימא תמאו , תורוקמ רפסמ קודב , מידקוממ מישופיה עצב
- ****grep****: - תינמזוב סיבר מיצבקב תוינבת שופיחל שמתשה . מירזוח מיסופד רותיאו רוקמ דוק חותינל ליעי הז ילכ

חוויד טמרופ

. תודבועב סיבוגמו , מירורב , סינבומ ויהי ללש תוח"וד
, תונמוסמ תויזכרמ תודוקנ , תורורב תורתוכב שמתשה
. סינימא תורוקממ סיטוטיצו

4.2.2 הבנת השינויים

נבחן את השינויים המהותיים שביצענו:

הוספת רשימת כלים: הוספנו את השדה tools כרשימה עם שלושה כלים. שימו לב שהסדר אינו משנה - הסוכן יחליט באופן דינמי באיזה כלי להשתמש בהתאם להקשר.

הרחבת ה-system_prompt: הוספנו מדור "שימוש בכלים" המנחה את הסוכן כיצד ומתי להשתמש בכל כלי. הנחיות אלו קריטיות - הן מגדירות את ה"תרבות הארגונית" של הסוכן ביחס לשימוש בכלים.

דגש על דיוק: ההנחיה "אל תשער או תמציא מידע" חיונית במיוחד כאשר הסוכן יכול לחפש באינטרנט. אנו רוצים שהסוכן יסמוך על מקורות אמינים ולא יזייף מידע.

4.2.3 דוגמה לשימוש מעשי

נניח שאנו מבקשים מהסוכן expert-analyst לנתח את השימוש בפונקציה מסוימת בפרויקט. הנה תרחיש האינטראקציה:

```
# Invoke the expert-analyst agent
$ claude agent run expert-analyst
```

```
# User query
```

User: טקירופב calculate_total היצקנופב מישומישה לכ תא אצמ אנא
, תשמשמ איה מיצבק וליאב מכסו ,

```
# Agent workflow (internal reasoning):
```

```
# 1. Uses grep to search for "calculate_total" pattern
# 2. Analyzes results to identify all files
# 3. Uses read on relevant files to understand context
# 4. Compiles a structured report
```

יתעצב:() מרוי Agent

:מיאצממה הנהו טקיוורפה לש הפיקמ הקירס

```
## calculate_total היצקנופב מישומיש מוכיס
```

```
### ואצמנואצמנש מיצבק
```

:מיצבק 4 תרגסמב היצקנופב מישומיש 7

1. ****src/billing/invoices.py**** (מישומיש 3)
 - תינובשה ךס בושיה: 45 הרוש
 - טירפ רובע יקלח מוכס בושיה: 102 הרוש
 - סמ ינפל ללוק מוכס בושיה: 156 הרוש
2. ****src/reports/monthly_summary.py**** (מישומיש 2)
 - רבטצמ ישדוה מוכס בושיה: 78 הרוש
 - חוקל יפל עצוממ בושיה: 134 הרוש
3. ****tests/test_billing.py**** (שומיש 1)
 - בושיה תוניקת תקידב: 23 הרוש
4. ****src/analytics/metrics.py**** (שומיש 1)
 - תוסנכה תקירטמ בושיה: 67 הרוש

```
### היצקנופהתורעה
```

ץלמומ.סיחווידהו בויהה ילודומב רקיעב תשממש
טלקה טמרופ תניחבמ סיבקע מישומישה לכש אדוול

בדוגמה זו, הסוכן השתמש בכלי grep למציאת כל המופעים של הפונקציה, ובכלי read לקריאת ההקשר של כל שימוש. התוצאה היא דו"ח מסודר ומקצועי.

4.2.4 הרחבה עתידית: כלים נוספים

בעתיד, ניתן יהיה להוסיף כלים נוספים בהתאם לצורך:

tools:

- read

- websearch
- grep
- glob # For file pattern matching
- webfetch # For fetching specific URLs
- write # For creating new files (use carefully!)
- edit # For editing existing files

חשוב לזכור שככל שמוסיפים כלים, יש להרחיב גם את ההנחיות ב-system_prompt כדי להבהיר כיצד להשתמש בהם באופן אחראי.

4.3 עקרון ההרשאה המינימלית (Principle of Least Privilege)

עקרון זה, שמקורו בעולם אבטחת המידע, גורס שכל ישות (משתמש, תהליך, או במקרה שלנו - סוכן AI) צריכה לקבל רק את ההרשאות המינימליות הנדרשות לביצוע תפקידה. יישום עקרון זה בהקשר של כלים לסוכנים מהווה הגנה חיונית מפני שימוש לרעה מכוון או בשוגג.

4.3.1 סיכוני הרשאות מופרזות

נניח שהענקנו לסוכן ניתוח לוגים את כל הכלים הקיימים, כולל edit, write, ו-bash. במקרה של באג במודל או הנחיה שגויה מהמשתמש, הסוכן עלול:

- למחוק או לשנות קבצים קריטיים
- להריץ פקודות מערכת מסוכנות
- לחשוף מידע רגיש לאינטרנט
- לגרום לעומס יתר על המערכת

אילו הסוכן היה מוגבל רק לכלים read ו-grep, הנזק האפשרי היה מצומצם לחשיפת מידע (שגם כך חמורה), אך לא היה כולל שינוי או השמדה של נתונים.

4.3.2 מטריצת כלים לפי סוגי סוכנים

טבלה 2 ממחישה כיצד עקרון ההרשאה המינימלית מתורגם להגדרת כלים עבור סוגי סוכנים שונים:

כפי שניתן לראות, הסוכן expert-analyst שלנו (מסוג Research Agent) זקוק לכלי קריאה וחיפוש, אך לא לכלי כתיבה או הרצה. זהו בדיוק המינימום הנדרש לתפקידו.

4.3.3 אסטרטגיית הוספה הדרגתית

גישה מומלצת היא להתחיל עם מינימום כלים ולהוסיף כלים נוספים רק כאשר מתגלה צורך ממשי. תהליך זה דומה לפיתוח איטרטיבי:

Table 2: Recommended Tool Matrix by Agent Role

Agent Type	read	websearch	grep	write/edit	bash
Log Analyzer	✓	×	✓	×	×
Research Agent	✓	✓	✓	×	×
Code Generator	✓	✓	✓	✓	×
DevOps Agent	✓	×	✓	✓	✓(limited)
Documentation Writer	✓	✓	✓	✓	×

1. גרסה ראשונית: הגדירו את הסוכן עם רק כלי read

2. בדיקה: הריצו את הסוכן על מקרי בוחן טיפוסיים

3. זיהוי חסר: האם הסוכן מבקש מידע שאינו נגיש לו?

4. הוספה מבוקרת: הוסיפו כלי ספציפי (למשל websearch)

5. תיעוד: רשמו מדוע הכלי הנוסף נדרש

6. חזרה לשלב 2: המשיכו לבדוק

גישה זו מבטיחה שכל כלי מוצדק ותועד, ומונעת "אינפלציית הרשאות" עם הזמן.

4.4 אימות והרצה: בדיקת הסוכן המורחב

לאחר שהוספנו את הכלים, חיוני לבדוק שהסוכן אכן יכול לגשת אליהם ומשתמש בהם כראוי.

4.4.1 בדיקת תקינות הגדרה

לפני ההרצה הראשונה, נוודא שקובץ ה-YAML תקין:

```
# Validate YAML syntax
python -c "import yaml; yaml.safe_load(open('expert-analyst.yaml'))"

# Check agent definition
claude agent list

# Expected output should include:
# - expert-analyst
# Description: ... מיונת תפילשל דעוימה החמום נכוס
# Model: claude-sonnet-4-5-20250929
# Tools: read, websearch, grep
```

4.4.2 בדיקת כלי read

ניצור קובץ בדיקה ונבקש מהסוכן לקרוא אותו:

```
# Create test file
cat > test_document.md << 'EOF'
# Test Document for expert-analyst
```

This is a **test** document to verify the **read** tool.

Key Information:

- Project: Knowledge-Agent-Yoram
- Purpose: Testing built-in tools
- Status: In development

EOF

```
# Run agent with read test
claude agent run expert-analyst
```

```
# User query inside agent session:
```

```
# "וב מושר המ יל רפסו test_document.md צבוקה תא "ארק"
```

```
# Expected: Agent should successfully read and summarize the file
```

4.4.3 בדיקת כלי websearch

נבדוק שהסוכן יכול לחפש מידע עדכני:

```
# Inside agent session:
```

```
User: Anthropic – ידילע המסרופש Claude API לש הנורחאה הסרגה יהמ
```

```
# Expected: Agent should use websearch to find current information
# and provide an accurate answer with source attribution
```

4.4.4 בדיקת כלי grep

נבדוק יכולת חיפוש דפוסים:

```
# Create test Python files
mkdir -p test_project
cat > test_project/main.py << 'EOF'
def calculate_total(items):
```

```
    return sum(item.price for item in items)
EOF
```

```
cat > test_project/utils.py << 'EOF'
def calculate_total_with_tax(items, tax_rate):
    subtotal = calculate_total(items)
    return subtotal * (1 + tax_rate)
EOF
```

Inside agent session:

User: `test_project` הייקיב `calculate_total` היצקנופכ מישומישה לכ תא אצמ

Expected: Agent uses grep to find both occurrences and reports them

4.5 פתרון בעיות נפוצות

במהלך העבודה עם כלים מובנים, עשויות להתעורר מספר בעיות טיפוסיות.

4.5.1 בעיה: הסוכן אינו משתמש בכלים

תסמינים: הסוכן עונה רק מתוך ידע פנימי ולא משתמש ב-`read` או `websearch` גם כאשר נדרש.

סיבות אפשריות:

- ה-`system_prompt` אינו מנחה את הסוכן להשתמש בכלים

- השאלה מנוסחת באופן שאינו מעורר צורך ברור בכלי חיצוני

- הכלים לא הוגדרו נכון ברשימת `tools`

פתרון: הוסיפו הנחיה מפורשת ב-`system_prompt`: "עליך להשתמש בכלים הזמינים לך לאיסוף מידע. אל תסתמך רק על ידע פנימי."

4.5.2 בעיה: שגיאת הרשאות בעת שימוש ב-`read`

תסמינים: הסוכן מדווח "Permission denied" בעת ניסיון לקרוא קובץ.

סיבות: הקובץ מוגן בהרשאות מערכת שאינן מאפשרות למשתמש המריץ את הסוכן לקרוא אותו.

פתרון: שנו את הרשאות הקובץ או הריצו את הסוכן עם משתמש בעל הרשאות מתאימות:

```
# Fix file permissions
chmod 644 restricted_file.txt
```

```
# Or run agent with appropriate user
sudo -u appropriate_user claude agent run expert-analyst
```

4.5.3 בעיה: websearch לא מחזיר תוצאות רלוונטיות

תסמינים: החיפוש מחזיר תוצאות לא קשורות או ישנות.

סיבות: השאלה מנוסחת באופן מעורפל מדי או כללי מדי.

פתרון: נסחו מחדש את השאלה להיות ספציפית יותר, או הדריכו את הסוכן לנסח שאילתת חיפוש טובה יותר:

```
system_prompt: |
    תופיצפס שופיה תותליאש חסנ , websearch ילכב שמתשמ רשאכ
    .תויטנוולר תוינכט חתפמ תולימב שמתשה .תוקיודמו
```

הבוט המגוד: "Claude_API_2025_latest_version_documentation"

הער המגוד: "Claude_information"

4.6 סיכום

בפרק זה הרחבנו את יכולות הסוכן expert-analyst על-ידי הוספת שלושה כלים מובנים חיוניים: read לקריאת קבצים, websearch לאיסוף מידע עדכני, ו-grep לחיפוש דפוסים. למדנו את עקרון ההרשאה המינימלית והבנו כיצד לבחור כלים באופן אחראי. בדקנו את תקינות ההגדרה והרצנו בדיקות פונקציונליות לכל כלי. הסוכן שלנו כעת מצויד ביכולות איסוף מידע בסיסיות, אך עדיין מוגבל לכלים סטנדרטיים. בפרק הבא, נעבור לשלב מתקדם יותר: בניית SKILL מותאם אישית שיאפשר לסוכן לבצע משימות מורכבות ומותאמות במיוחד לצרכי הפרויקט שלנו.

4.7 יכולות מותאמות אישית: בניית SKILL

כאשר אנו עוברים מעבר לכלים המובנים של Claude CLI, אנו נכנסים לתחום היצירתיות האמיתית בבניית סוכנים. SKILL הוא רכיב מודולרי שמאפשר לנו להגדיר יכולות ספציפיות למשימה, מורכבות ומותאמות אישית, החורגות מהפונקציונליות הגנרית של כלים בסיסיים כגון read או websearch.

המעבר מכלים מובנים ל-SKILL מותאם אישית הוא כמו המעבר משימוש בכלי עבודה סטנדרטי ליצירת כלי מיוחד בדיוק לצורך המשימה שלפניכם. זהו שלב קריטי בהפיכת הסוכן שלכם מביצוע משימות גנריות לביצוע תפקידים מקצועיים וספציפיים ברמה גבוהה.

4.7.1 הסבר עקרוני: הצורך ב-SKILL

בעוד שהכלים המובנים מספקים פונקציות גנריות חיוניות - קריאת קבצים, כתיבת טקסט, חיפוש באינטרנט - הם אינם מספקים כאשר אנו זקוקים ללוגיקת ביצוע מורכבת וספציפית למשימה. דמיינו שאתם זקוקים לעבד קובץ נתונים גולמי (raw data), לנתח אותו לפי קריטריונים מדויקים, ולהציג את התוצאה בפורמט מובנה ופורמלי - משימה כזו דורשת יותר מסתם קריאת קובץ.

כאן נכנס SKILL לתמונה. SKILL הוא יכולת מורחבת (extended capability) המוגדרת כתיקיה המכילה קובץ תיאור מרכזי בשם SKILL.md. קובץ זה משמש כ-manifest - מסמך הגדרה רשמי המתאר את המטרה, הקלט הנדרש והלוגיקה שעל הסוכן לבצע.

הבדלים מהותיים בין כלי מובנה ל-SKILL:

1. רמת הפירוט:

כלי מובנה כגון read פשוט קורא קובץ ומחזיר את תוכנו. SKILL יכול להגדיר תהליך רב-שלבי: קרא קובץ, נתח את המבנה שלו, זהה דפוסים, בצע חישובים סטטיסטיים, והחזר דוח מעוצב.

2. סכימת קלט מובנית:

SKILL מגדיר input_schema - מפרט פורמלי של הפרמטרים הנדרשים. לדוגמה, SKILL לניתוח נתונים יכול לדרוש file_path (נתיב לקובץ) ו-analysis_type (סוג הניתוח הנדרש: סטטיסטי, שכיחויות, חריגים). המודל יודע אוטומטית אילו ערכים לספק.

3. הוראות ביצוע מפורשות:

בתוך SKILL.md, אתם כותבים הוראות צעד-אחר-צעד ל-Claude, המנחות אותו כיצד לבצע את המשימה בדיוק. זה כמו לתת למודל תסריט מפורט במקום הנחיה כללית.

4. מודולריות ושימוש חוזר:

SKILL הוא רכיב עצמאי שניתן להשתמש בו בסוכנים שונים, לשתף עם צוות, ולתחזק בנפרד. הוא מהווה אבן בניין בספריית היכולות של הארגון שלכם.

דוגמה קונקרטית:

נניח שאתם בונים את הסוכן Yoram-Agent-Knowledge, סוכן מומחה לניהול ידע. אתם רוצים שהסוכן יוכל לעבד קובצי log או נתונים מופרדים (CSV-like), לנתח אותם, ולהציג דוח סיכום מובנה וטבלאי. כלי read יקרא את הקובץ, אך הוא לא יבצע את הניתוח והפורמט המבני. כלי grep יוכל לחפש תבניות, אך לא לסכם ולהציג דוח פורמלי. כאן

תיצרו SKILL בשם structured-report, שמטרתו לבצע את כל התהליך המורכב הזה כיכולת אחת מקיפה.

4.7.2 ההגדרה המעשית: יצירת תיקיית ה-SKILL וקובץ ה-SKILL.md

כעת נעבור מהתיאוריה לפרקטיקה. ניצור SKILL חדש בשם structured-report עבור הסוכן Yoram-Agent-Knowledge שלנו. SKILL זה יהיה מוקדש לעיבוד, ניתוח והצגה מבנית של נתונים גולמיים - לוגים, קבצי טקסט מופרדים, או כל מקור נתונים דומה - והפיכתם לדוח טבלאי פורמלי ומסודר.

שלב א': יצירת מבנה ה-SKILL

Claude CLI מצפה למצוא SKILLs בנתיב ספציפי בתוך תיקיית הפרויקט: skills/. בתוך תיקייה זו, כל SKILL מיוצג על ידי תיקייה נפרדת המכילה את קובץ המניפסט SKILL.md. נווטו לתיקיית הפרויקט Knowledge-Agent-Yoram/ ונבצע את הפעולות הבאות:

```
yrotceriD LLIKS gnitaerC - sdnammoC hsaB
```

```
# Navigate to project root
cd Knowledge-Agent-Yoram/

# Create new SKILL directory
mkdir skills/structured-report
```

המבנה כעת נראה כך:

```
erutcurtS yrotceriD
```

```
Knowledge-Agent-Yoram/|—
  agents/|
    └─ expert-analyst.yaml|—
  commands/|—
  knowledge/|—
  skills/
    └─ structured-report/ # <-- New SKILL directory
```

שלב ב': יצירת קובץ ה-SKILL.md Manifest

בתוך התיקייה structured-report/, נצור את הקובץ המרכזי SKILL.md. קובץ זה מחולק לשני חלקים עיקריים:

1. **Frontmatter ב-YAML:** מפרט את המטא-נתונים של ה-SKILL - שם, תיאור, וסכימת הקלט.

2. **גוף המסמך ב-Markdown:** הוראות מפורטות ל-Claude כיצד לבצע את ה-SKILL. ניצור את הקובץ:

```
cd Knowledge-Agent-Yoram/skills/structured-report/
touch SKILL.md
```

כעת נזין את תוכן ה-SKILL.md. הנה התוכן המלא של הקובץ:

snoitcurtsnI + rettamtnorF - tnetnoC dm.LLIKS

[illegible]

הסבר מפורט על המרכיבים:

א. Frontmatter - המטא-נתונים:

המקטע בין — הוא YAML frontmatter, והוא מכיל את ההגדרות הפורמליות של ה-SKILL:

- name: המזהה הייחודי של ה-SKILL. זה השם שבו תפנו אליו בקובץ ה-YAML של הסוכן (ברשימת tools).

- description: תיאור קצר ומדויק של מטרת ה-SKILL. המודל משתמש בתיאור זה כדי להחליט מתי להפעיל את ה-SKILL. תיאור טוב הוא קריטי - אם הוא מעורפל, המודל לא יבין מתי להשתמש ביכולת זו.

- input_schema: זהו המפרט הפורמלי של הקלט הנדרש. הוא מוגדר בתקן JSON Schema. כאן הגדרנו שני פרמטרים חובה:

- file_path: נתיב לקובץ הנתונים.

- analysis_type: סוג הניתוח הרצוי (למשל: "statistical", "frequency", "anomalies").

כאשר המודל מחליט להפעיל את ה-SKILL, הוא יודע אוטומטית לחלץ את הערכים הללו מהקשר השיחה (למשל, מבקשת המשתמש) וימלא אותם בפורמט הנכון.

ב. גוף ההוראות - המדריך לביצוע:

החלק השני של הקובץ (אחרי — השני) הוא המדריך המפורט ל-Claude כיצד לבצע את המשימה. זהו למעשה תסריט צעד-אחר-צעד שהמודל עוקב אחריו בדיוק. ההוראות כתובות ב-Markdown ומחולקות לשלבים ברורים.

למה זה חשוב? ללא הוראות מפורשות, המודל עלול לפרש את המשימה בצורה שונה מכוונתכם. הוראות ברורות מבטיחות ביצוע עקבי ומדויק.

שלב ג': חיבור ה-SKILL לסוכן

כעת, לאחר שיצרנו את ה-SKILL, עלינו לאפשר לסוכן Yoram-Agent-Knowledge להשתמש בו. נעדכן את קובץ ה-Agent Manifest שלנו: פתחו את הקובץ agents/expert-analyst.yaml והוסיפו את שם ה-SKILL לרשימת הכלים:

המשמעות המעשית:

כעת, כאשר משתמש מבקש מהסוכן משימה כגון Log-ה ז'בוקמ ינבמ יטסיטטס חוד קפה אנא" data-bbox="112 606 886 688">"data.log", המודל Opus יזהה שהתיאור (description) של ה-SKILL structured-report תואם למשימה זו, ויבחר להפעיל אותו באמצעות הפרמטרים המתאימים: לאחר מכן, המודל יבצע את ההוראות שב-SKILL.md:

1. ישתמש בכלי read לקריאת data.log.

2. יעבד את התוכן לפי ההוראות (ארגון לטבלה פנימית).

3. יבנה דוח טבלאי מסודר בהתאם לסוג הניתוח הנדרש.

4. יחזיר את הדוח המעוצב ב-Markdown למשתמש.

דוגמה מעשית להפעלת הסוכן עם ה-SKILL:

נניח שיצרנו קובץ דמוי CSV בשם transactions.csv:

נפעיל את הסוכן עם הבקשה הבאה:

תוצאה צפויה (באופן תיאורטי):

LLIKS htiw lmay.tsylana-trepixe detadpU

```
# Knowledge-Agent-Yoram/agents/expert-analyst.yaml
---
name: "expert-analyst"
description: "חותינן, תמדקתמטסינותננפתילשלטדעוימהחמומטלכוס"
קוידטלעטרומשלטביחממהזטלכוסט. יקמטעדימטוכיסוטסיינכטסיכמטמט
טלחומטיעוצקמט."
model: "opus"
tools:
  - read
  - websearch
  - grep
  - structured-report # <-- NEW: שדחה SKILL-ה תפטוה

system_prompt: |
  קמונמו תמואמ תויהל בייח טלפ לכ. קיודמו ריכב טסילנא התא
  .פלשנ וא קפוטש עדימה לע סטבתהב
  ,תיתטיש תויהל תבייח דלש השיגה. תויצלוקפטמ ענמיה
  .תימשרו תיעוצקמ
```

elpmaxE noitacovnI LLIKS

```
# The model automatically determines:
file_path = "data.log"
analysis_type = "statistical"

# And invokes the SKILL with these parameters
```

vsc.snoitcasnart - eliF ataD elpmaS

```
TransactionID,Amount,Status
TX001,450.00,SUCCESS
TX002,120.00,FAILED
TX003,98.50,SUCCESS
TX004,305.75,SUCCESS
TX005,89.00,FAILED
```

dnammoC noitacovnI tnegA

```
claude agent run expert-analyst --task "לעהנבומטיטטיטטחותיננעצב"
"יעוצקמחודקפהוט'transactions.csv'טבוקבשטיימלוגהסינותנה
```

המודל יזהה שהמשימה דורשת עיבוד נתונים מובנה, ויפעיל את ה-SKILL structured--report

1. יקרא את transactions.csv באמצעות read.

2. יעבד את הנתונים (יזהה 5 טרנזקציות, 3 מוצלחות, 2 כושלות).

3. יבנה דוח טבלאי פורמלי ומסודר ב-Markdown.

4. יחזיר תשובה מקצועית ורשמית (בהתאם ל-system_prompt).

יתרונות השימוש ב-SKILL:

1. **עקביות:** כל פעם שהסוכן מבצע משימת ניתוח נתונים, הוא עוקב אחרי אותו תסריט מדויק. אין שונות בין ביצועים.

2. **שימוש חוזר:** אתם יכולים להשתמש ב-SKILL זה בסוכנים אחרים פשוט על ידי הוספתו לרשימת tools שלהם.

3. **תחזוקה קלה:** אם אתם רוצים לשפר את לוגיקת הניתוח, אתם מעדכנים רק את SKILL.md, ואין צורך לשנות את קובצי הסוכנים עצמם.

4. **ספריית יכולות ארגונית:** ככל שאתם בונים יותר SKILLS, אתם יוצרים ספריית יכולות רב-שימושית שכל הצוות יכול להשתמש בה.

שיקולים חשובים בבניית SKILL

א. **תיאור ברור ומדויק:** ה-description חייב לתאר במדויק **מתי** להשתמש ב-SKILL. תיאור מעורפל יגרום למודל לא להבין מתי להפעיל את היכולת.

ב. **סכימת קלט מובנית:** הגדירו בבירור אילו פרמטרים נדרשים ומה התפקיד של כל אחד. השתמשו ב-JSON Schema תקני.

ג. **הוראות ברורות וצעד-אחר-צעד:** ככל שההוראות ב-SKILL.md יותר מפורטות וברורות, כך הביצוע יהיה מדויק יותר.

ד. **בדיקה ואימות:** לאחר יצירת ה-SKILL, הריצו אותו עם מקרי בוחן שונים כדי לוודא שהוא עובד כצפוי.

סיכום ביניים:

בפרק זה למדנו את המושג SKILL - יכולת מותאמת אישית המרחיבה את הסוכן מעבר לכלים המובנים. יצרנו SKILL בשם structured-report עבור הסוכן Yoram-Agent-Knowledge, כולל:

- תיקיית SKILL בנתיב skills/structured-report/ - קובץ מניפסט SKILL.md עם frontmat-ter ב-YAML והוראות ב-Markdown - סכימת קלט מובנית (input_schema) - הוראות ביצוע מפורטות - חיבור ה-SKILL לסוכן באמצעות קובץ ה-YAML

בפרק הבא (פרק 6 - Python אינטגרציה חזקה), נרחיב את ה-SKILL שלנו על ידי הוספת קוד Python שיבצע חישובים סטטיסטיים מורכבים, ובכך נהפוך את הסוכן שלנו לכלי ניתוח נתונים אמיתי ועוצמתי.

בשלב זה: הסוכן Yoram-Agent-Knowledge מסוגל לקרוא קבצים, לחפש באינטרנט, לבצע grep, **ולהפעיל את ה-SKILL המותאם אישית structured-report**. זהו צעד משמעותי בהתפתחות הסוכן שלנו.

6 אינטגרציה חזקה: הוספת קוד Python

6.1 הסבר עקרוני: הרחבת ה-SKILL באמצעות קוד

בפרקים הקודמים למדנו כיצד לבנות סוכן AI עם כלים מובנים ויכולות מותאמות אישית דרך Skills. אולם, ככל שאנו מנסים לבנות סוכנים מתקדמים יותר, אנו נתקלים במגבלה מהותית: מודלי שפה, גם המתקדמים שבהם, אינם מתוכננים לבצע חישובים מתמטיים מדויקים, עיבוד נתונים סטטיסטי מורכב, או אינטראקציה ישירה עם ספריות חיצוניות כגון pandas או numpy.

כאשר אנו מבקשים מסוכן לבצע ניתוח סטטיסטי על אלפי שורות נתונים, לחשב מטריצות מתמטיות, או לעבד קבצי CSV גדולים, המודל יתקשה לספק תוצאות מדויקות. זו לא חולשה של המודל, אלא פשוט אי התאמה בין התפקיד שלו – יצירת טקסט והבנת הקשר – לבין משימות חישוביות שדורשות דיוק מוחלט. למעשה, ניסיון להעמיס על מודל שפה משימות חישוביות דומה לבקשה ממוזיקאי מחונן לבנות גשר: הכלים והמיומנות אינם מתאימים למשימה.

פה נכנס לתמונה העיקרון של **אינטגרציה חזקה** (Strong Integration): שילוב קוד תכנות מסורתי – בדרך כלל Python בשל הפופולריות והספריות העשירות שלו – עם היכולות הטקסטואליות והאנליטיות של הסוכן. במודל זה, הסוכן משמש כשכבת תיזמור חכמה: הוא מזהה מתי נדרש עיבוד חישובי, מפעיל את קוד ה-Python המתאים, מקבל את התוצאות המדויקות, ומשתמש בהן כדי לנסח תשובה מקצועית ומפורטת למשתמש.

ארכיטקטורה זו מאפשרת לנו לבנות סוכנים היברידיים: הם משלבים את החזקה של AI שפתי – הבנת הקשר, ניתוח טקסט חופשי, ניסוח תשובות טבעיות – עם הדיוק והמהירות של קוד קומפילד. התוצאה היא סוכן המסוגל לקחת שאלה כמו "אנא נתח את המגמות החודשיות במכירות ושלח לי דוח מפורט", לעבד את הנתונים באמצעות Python, ולהציג ניתוח טקסטואלי מעמיק המבוסס על תוצאות חישוביות מדויקות.

6.1.1 מבנה ה-Skill המשולב

עבור סוכן "expert-analyst" שלנו, נרחיב את ה-SKILL בשם structured-report שיצרנו בפרק הקודם. במקום שהסוכן ינסה לנתח נתונים באופן טקסטואלי בלבד, נוסיף תיקיית scripts/ בתוך ה-SKILL, ובה נשמור סקריפט Python בשם data_processor.py. סקריפט זה יבצע את העבודה החישובית הכבדה: קריאת קבצי CSV, חישוב סטטיסטיקות (ממוצע, חציון, סטיית תקן), זיהוי חריגות, ויצירת פלט מובנה בפורמט JSON.

כאשר המשתמש יבקש ניתוח, הסוכן יפעיל את ה-SKILL, אשר בתורו יקרא לסקריפט ה-Python עם הפרמטרים הרלוונטיים, יקבל את התוצאות, וישתמש בהן כדי לנסח דוח מקצועי ומפורט. כל זה מתבצע בצורה שקופה למשתמש, המקבל תשובה אחת מקיפה ומדויקת.

6.2 ההגדרה המעשית: יצירת התשתית

6.2.1 ארגון תיקיית ה-scripts

השלב הראשון הוא ליצור מבנה תיקיות מסודר. עבור ה-SKILL שלנו, נרחיב את המבנה הקיים:

Listing 7: Python עם SKILL יצירת מבנה תיקיות ל-

```
# Navigate to the Skill directory
cd Knowledge-Agent-Yoram/ skills / structured-report /

# Create scripts folder
mkdir scripts

# Create the Python processor script
touch scripts/data_processor.py
```

מבנה התיקיות המלא של ה-SKILL כעת נראה כך:

```
Knowledge-Agent-Yoram/
├── skills/
│   ├── structured-report/
│   │   ├── SKILL.md           # Manifest and instructions
│   │   └── scripts/
│   │       └── data_processor.py # Python computational backend
```

6.3 עיבוד נתונים וניתוח: סקריפט ה-Python

כעת נבנה את הסקריפט המרכזי `data_processor.py`. סקריפט זה ידגים עקרונות חשובים:

1. **הגבלת 150 שורות:** זוהי דרישה קריטית מתוך ההנחיות הגלובליות של הפרויקט. קוד תמציתי יותר קל לתחזוקה, לבדיקה ולהבנה.

2. **ממשק נקי:** הפונקציה המרכזית `analyze_data()` מקבלת נתונים גולמיים וסוג ניתוח, ומחזירה מילון Python עם תוצאות מובנות.

3. **טיפול בשגיאות:** כל קוד ייצורי חייב להכיל טיפול מתאים בחריגות.

4. **תיעוד ברור:** תגובות ב-`docstrings` באנגלית, עם הערות בעברית היכן שמתאים.

— סקריפט עיבוד נתונים (130 שורות) data_processor.py: Listing 8

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Data_processor_for_statistical_analysis-Chapter_06_Python_Integration.
Max_Lines: 150 (MANDATORY-from global CLAUDE.md)
"""

import json
from typing import Dict, List
import statistics

def analyze_data(raw_data: str, analysis_type: str) -> Dict:
    """
    Perform structured analysis on raw data and return summary.

    Args:
    raw_data (str): Raw input data (CSV-like or text lines)
    analysis_type (str): Type of analysis requested
    - "statistical": Basic statistical summary
    - "frequency": Frequency distribution
    - "outlier": Detect outliers using IQR method

    Returns:
    Dict: Structured analysis results
    """
    try:
        lines = raw_data.strip().split('\n')
        num_records = len(lines)
        if num_records == 0:
            return {"error": "No data provided", "status": "failed"}
        if analysis_type == "statistical":
            return _statistical_analysis(lines)
        elif analysis_type == "frequency":
            return _frequency_analysis(lines)
        elif analysis_type == "outlier":
            return _outlier_detection(lines)
        else:
```



```

        return {
            "num_records": num_records,
            "analysis_type": f"Unknown: {analysis_type}",
            "status": "partial",
            "message": "Analysis type not fully implemented"
        }
    except Exception as e:
        return {"error": str(e), "status": "failed"}

def _statistical_analysis(lines: List[str]) -> Dict:
    """Compute basic statistical measures."""
    num_records = len(lines)
    avg_length = sum(len(line) for line in lines) / num_records
    numeric_values = []
    for line in lines[1:]:
        parts = line.split(',')
        for part in parts:
            try:
                numeric_values.append(float(part.strip()))
            except ValueError:
                continue
    result = {
        "num_records": num_records,
        "avg_line_length": round(avg_length, 2),
        "analysis_type": "Statistical Summary",
        "status": "completed"
    }
    if numeric_values:
        result["numeric_stats"] = {
            "count": len(numeric_values),
            "mean": round(statistics.mean(numeric_values), 2),
            "median": round(statistics.median(numeric_values), 2),
            "std_dev": round(statistics.stdev(numeric_values), 2)
            if len(numeric_values) > 1 else 0,
            "min": min(numeric_values),
            "max": max(numeric_values)
        }
    return result

```

```

def _frequency_analysis(lines: List[str]) -> Dict:
    """Analyze frequency distribution of values."""
    frequency_map = {}
    for line in lines:
        line_stripped = line.strip()
        if line_stripped:
            frequency_map[line_stripped] = \
                frequency_map.get(line_stripped, 0) + 1
    sorted_freq = sorted(
        frequency_map.items(),
        key=lambda x: x[1],
        reverse=True
    )
    return {
        "num_records": len(lines),
        "unique_values": len(frequency_map),
        "top_5_frequent": sorted_freq[:5],
        "analysis_type": "Frequency Distribution",
        "status": "completed"
    }

def _outlier_detection(lines: List[str]) -> Dict:
    """Detect outliers using Interquartile Range (IQR) method."""
    numeric_values = []
    for line in lines[1:]:
        parts = line.split(',')
        for part in parts:
            try:
                numeric_values.append(float(part.strip()))
            except ValueError:
                continue
    if len(numeric_values) < 4:
        return {
            "error": "Insufficient numeric data for outlier detection",
            "status": "failed"
        }
    q1 = statistics.quantiles(numeric_values, n=4)[0]
    q3 = statistics.quantiles(numeric_values, n=4)[2]
    iqr = q3 - q1

```

```

lower_bound = q1 - 1.5 * iqr
upper_bound = q3 + 1.5 * iqr
outliers = [v for v in numeric_values
             if v < lower_bound or v > upper_bound]
return {
    "num_records": len(numeric_values),
    "q1": round(q1, 2),
    "q3": round(q3, 2),
    "iqr": round(iqr, 2),
    "lower_bound": round(lower_bound, 2),
    "upper_bound": round(upper_bound, 2),
    "outliers_count": len(outliers),
    "outliers": outliers[:10],
    "analysis_type": "Outlier_Detection_(IQR)",
    "status": "completed"
}

```

שימו לב למספר היבטים קריטיים בקוד:

- הסקריפט הוא 130 שורות – מתחת למגבלה של 150.
- הפונקציה `analyze_data()` היא הממשק הציבורי היחיד. פונקציות העזר `_status_tical_analysis()`, `_frequency_analysis()` ו-`_outlier_detection()` הן פרטיות (מוסכמת Python עם קו תחתון).
- כל פונקציה מחזירה מילון Python עם מבנה עקבי, הכולל שדה `status` לבדיקת הצלחה.
- הקוד מטפל בשגיאות בצורה מסודרת ומחזיר הודעות שגיאה ברורות.

6.4 זרימת העבודה של אינטגרציה Python-SKILL

כעת שיש לנו סקריפט Python פונקציונלי, עלינו לעדכן את קובץ ה-`SKILL.md` כדי שהסוכן יידע כיצד להפעיל אותו. העדכון כולל הוספת הוראות מפורשות לקריאה לקוד.

6.4.1 עדכון SKILL.md

Listing 9: Python עם הפניה ל-`SKILL.md` עדכון קובץ

```

# Knowledge-Agent-Yoram / skills / structured-report / SKILL.md
---
name: "structured-report"
description: "מינותנלשתינבמגהצוהותינ, דוביעלשמשמ"
             תילמרופמוכיסתלבטל (מידרפומטסקטציבק, ) סיגולסיימלוג

```

```

מיימלוג מינותנ שישכ לייעפהל שי . הנבומ חוד ווא
. תימשר הגצהו ווגרא מישרודה
input_schema:
  type: object
  properties:
    file_path:
      type: string
      description: "חתנל שיש ימלוגה מינותנה קבוקל ביתנה ."
    analysis_type:
      type: string
      description: "לשמל (statistical , frequency , outlier)."
    required: [file_path , analysis_type]
---

## הנבומ חוד עוציבל תוארוה:

**1. נכותהתא וועטל ידכ `read` ילכב שמתשה: **מינותנ טלק
`file_path` - ב לבקתהש קבוקה ביתנמ אלמה

-ה תאו ארקנש נכותה תא רבעה: ** (Python) ימינפ דוביע
`analyze_data()` היצקנופה לא `analysis_type`
**`scripts/data_processor.py`**

**3. טפירקסהמילרוטקורטסה טלפה תא לבק: **חוד תקפהו חותינ
ילמרופיאלבט חוד תונבל ידכ הלא מינותנב שמתשה .
.סייטנוולרה מיאצממה תא שיגדמה סכסמו

**4. שמתשמל Markdown -ב בצועמה ינבמה ח"ודה תא רזחה: **יפוס טלפ
.

```

עדכון זה מבהיר לסוכן את השלבים המדויקים: קרא קובץ, הפעל Python, קבל תוצאות, נסח דוח. במערכות Claude CLI מתקדמות, קיימים מנגנונים אוטומטיים להפעלת סקריפטים על בסיס input_schema, אך כאן אנו מדגימים את ההכוונה הפורמלית.

6.4.2 חיבור ה-SKILL לסוכן

לבסוף, נוודא שהסוכן expert-analyst יכול להשתמש ב-SKILL המעודכן. אם עדיין לא עשינו זאת בפרק הקודם, נעדכן את agents/expert-analyst.yaml:

SKILLעדכון הסוכן להכיל את ה-: Listing 10

```
# Knowledge-Agent-Yoram/agents/expert-analyst.yaml
---
name: "expert-analyst"
description: "חותינ, תמדקתמ, סינותנ, תפילשל, דעוימה, החמומ, נכוס"
רומשל, ביוחמ, הז, נכוס. ריקמ, עדימ, סוכיס, סיינכט, סיכמסמ,
". טלחומ, יעוצקמ, קויד, לע
model: "opus"
tools:
  - read
  - websearch
  - grep
  - structured-report # SKILL with Python backend

system_prompt: |
  קמונמו תמואמ תויהל בייח טלפ לכ. קיודמו ריכב טסילנא התא
  .פלשנ וא קפוסש עדימה לע ססבתהב
  , תיתטיש תויהל תבייה קלש השיגה. תויצלוקפסמ ענמיה
  . תימשרו תיעוצקמ
```

6.5 הוספת ה-`data_processor.py` ל-`structured-report`

כעת, כאשר משתמש יפנה לסוכן `expert-analyst` ויבקש ניתוח נתונים, הזרימה המלאה תהיה:

1. **קבלת הבקשה:** המשתמש שולח: "בצע ניתוח סטטיסטי מובנה על הנתונים הגולמיים שבקובץ `data.csv`."

2. **זיהוי כוונה:** המודל `Opus` מזהה שהמשימה מתאימה לתיאור של `structured-report` `SKILL`.

3. **הפעלת SKILL:** הסוכן מפעיל את ה-`SKILL` עם הפרמטרים:

`file_path: "data.csv"` -

`analysis_type: "statistical"` -

4. **קריאת קובץ:** ה-`SKILL` משתמש בכלי `read` לקריאת `data.csv`.

5. **הפעלת Python:** ה-`SKILL` מפעיל את `data_processor.py` עם הנתונים הגולמיים וסוג הניתוח.

6. **עיבוד חישובי:** הסקריפט `Python` מבצע חישובים סטטיסטיים ומחזיר `JSON` מובנה.

7. **קבלת תוצאות:** ה-`SKILL` מקבל את ה-`JSON` עם הסטטיסטיקות.

8. **ניסוח דוח:** הסוכן משתמש בנתונים המדויקים לניסוח דוח מקצועי ב-Markdown.

9. **החזרה למשתמש:** המשתמש מקבל דוח מפורט הכולל סטטיסטיקות מדויקות בניסוח טבעי.

זרימה זו משלבת את החוזקות של שני העולמות: הדיוק החישובי של Python והיכולת הטקסטואלית של Claude.

6.6 סיכום והמשך

בפרק זה למדנו כיצד להרחיב את יכולות הסוכן באמצעות אינטגרציה חזקה עם קוד Python. יצרנו סקריפט עיבוד נתונים מלא תוך שמירה על מגבלת ה-150 שורות, עדכנו את ה-SKILL להפנות אליו, והדגמנו את הזרימה המלאה מבקשת משתמש ועד תשובה מבוססת חישובים. העיקרון המרכזי שלמדנו הוא שסוכני AI מתקדמים אינם צריכים לבצע הכל בעצמם. על ידי יצירת ארכיטקטורה היברידית, אנו מאפשרים לכל רכיב לבצע את המשימות שהוא עושה בצורה הטובה ביותר: ה-LLM מנתח הקשר ומנסח טקסט, ו-Python מבצע חישובים מדויקים.

בפרק הבא נמשיך להרחיב את הסוכן עם יכולות זיכרון וקטורי (RAG), שיאפשרו לו לגשת למאגרי ידע נרחבים מעבר לגבולות חלון ההקשר. הסוכן Yoram-Agent-Knowledge שלנו ממשיך להתפתח מכלי פשוט לישות אנליטית מתוחכמת.

חלק שלישי: תכונות מתקדמות

7 זיכרון וקטורי (RAG): ניהול ידע מתקדם

בפרקים הקודמים למדנו כיצד לבנות סוכנים אוטונומיים המסוגלים לבצע משימות מורכבות באמצעות כלים חיצוניים. עם זאת, כל סוכן שבנינו עד כה סובל ממגבלה מהותית: הוא מוגבל לידע שנמצא בתוך חלון ההקשר שלו. מהי משמעות המגבלה הזו? כיצד ניתן להתגבר עליה? והאם קיים מנגנון שיאפשר לסוכנים שלנו לגשת לכמויות עצומות של מידע מבלי להעמיס על חלון ההקשר?

בפרק זה נחקור את עולם ה-RAG (Retrieval-Augmented Generation), טכנולוגיה שמשנה באופן יסודי את היכולות של מודלי שפה גדולים. נלמד כיצד זיכרון וקטורי מאפשר לסוכנים לגשת למאגרי ידע עצומים, להבין את המשמעות הסמנטית של שאילתות, ולהחזיר מידע רלוונטי במדויק. נראה כיצד הפרוטוקול MCP (Model Context Protocol) משמש כגשר בין הסוכן למאגר הידע, ונבנה יחד סוכן מתקדם המסוגל לנצל את כל היכולות הללו.

7.1 הסבר עקרוני: חריגה מגבולות ההקשר

כל מודל שפה גדול פועל עם מגבלה טכנית מהותית הנקראת "חלון הקשר" (Context Window). זהו מספר הטוקנים המקסימלי שהמודל יכול לעבד בו-זמנית. לדוגמה, המודל Claude Sonnet 4.5 מסוגל לעבד עד 200,000 טוקנים בחלון הקשר אחד. נשמע מרשים, אך בואו נבחן מה המשמעות המעשית של מספר זה.

טוקן הוא יחידת הטקסט הבסיסית שהמודל מעבד. במילים אנגליות, טוקן בדרך כלל שווה למילה אחת או חלק ממילה. בעברית, בשל המורכבות של השפה והקידומות והסיימות, מילה אחת עשויה להתפרק למספר טוקנים. כלל אצבע סביר הוא ש-1,000 מילים שוות ערך לכ-1,300 טוקנים בטקסט מעורב עברית-אנגלית.

כעת נעריך: ספר טכני ממוצע מכיל כ-70,000 מילים, מה שמתורגם לכ-90,000 טוקנים. זאת אומרת שחלון ההקשר של Claude Sonnet 4.5 יכול להכיל כשני ספרים טכניים שלמים. נשמע מספיק? לא בהכרח. נניח שאנחנו בונים סוכן שמתמחה במתן ייעוץ טכני בתחום הפיננסים. הסוכן צריך לדעת את כל התקנות הרגולטוריות הרלוונטיות, מאות מסמכים; את ההיסטוריה הפיננסית של החברה, שנים של דוחות; את מאגר המאמרים האקדמיים בתחום, אלפי מחקרים; ואת תיעוד המוצרים והשירותים הפיננסיים, מאות מסמכים טכניים.

סך הכל, אנחנו מדברים על מיליוני טוקנים של מידע. אין שום דרך להכניס את כל המידע הזה לחלון ההקשר של המודל. זו בדיוק הבעיה ש-RAG בא לפתור. הרעיון המרכזי הוא פשוט אך מהפכני: במקום להעמיס את כל המידע על המודל, אנחנו מאחסנים אותו במאגר חיצוני ומביאים רק את החלקים הרלוונטיים בדיוק כשהם נחוצים. זה דומה לאופן שבו אנחנו, בני האדם, עובדים - איננו זוכרים כל מידע שאי-פעם למדנו, אלא יודעים היכן לחפש אותו כשאנחנו צריכים אותו.

הדוגמה הקונקרטית הבאה מבהירה את הבעיה: ארגון בינלאומי מחזיק 500 דוחות טכניים, כל אחד באורך ממוצע של 10,000 מילים. זה מסתכם ב-5 מיליון מילים, שמתורגמים לכ-6.5 מיליון טוקנים. אפילו אם היה לנו מודל שיכול לעבד כמות כזו בבת אחת, העלות הכלכלית

הייתה אסטרונומית. במחירים הנוכחיים של Claude Sonnet 4.5, עיבוד 6.5 מיליון טוקנים בכל שאילתה היה עולה כמה עשרות דולרים לשאילתה בודדת. זה לא בר-קיימא כלכלית, וגם לא נדרש - רוב השאילתות זקוקות רק לאחוזים בודדים מהידע הזה.

7.2 עקרונות Retrieval-Augmented Generation

Retrieval-Augmented Generation, או בקיצור RAG, הוא ארכיטקטורה שמשלבת שני רכיבים מרכזיים: מנגנון חיפוש (Retrieval) ומודל שפה יוצר (Generation). הרעיון, שפורסם לראשונה במאמר פורץ דרך של Lewis ועמיתיו ב-2020 [9], שינה לחלוטין את הגישה לבניית מערכות המבוססות על מודלי שפה גדולים.

תהליך ה-RAG מתחלק לארבעה שלבים עיקריים, כל אחד קריטי להצלחת המערכת כולה.

7.2.1 שלב האינדקס: הכנת מאגר הידע

בשלב הראשון, אנחנו לוקחים את כל המסמכים שברשותנו ומפרקים אותם ליחידות קטנות יותר, הנקראות "קטעים" (chunks). כל קטע מכיל כמה פסקאות של טקסט, בדרך כלל בין 200 ל-1000 מילים. למה לא לשמור כל מסמך כמקשה אחת? כי אנחנו רוצים לאחזר רק את החלקים הרלוונטיים מכל מסמך, לא את כולו. אם מסמך של 50 עמודים דן בעשרה נושאים שונים, ואנחנו שואלים שאלה על נושא אחד מהם, אנחנו רוצים רק את הקטע הרלוונטי, לא את כל המסמך.

לאחר שחילקנו את המסמכים לקטעים, אנחנו ממירים כל קטע לייצוג מתמטי הנקרא "הטמעה" (embedding). הטמעה היא וקטור של מספרים, בדרך כלל בין 768 ל-1536 מימדים, המייצג את המשמעות הסמנטית של הטקסט. הרעיון המבריק כאן הוא שטקסטים בעלי משמעות דומה יקבלו וקטורים דומים במרחב הווקטורי. לדוגמה, המשפטים "מודלי שפה גדולים הם כלי עוצמתי" ו-"LLMs are powerful tools" יקבלו וקטורים קרובים מאוד זה לזה, למרות שהם בשפות שונות.

את ההטמעות הללו אנחנו שומרים במאגר וקטורי מיוחד, כמו Qdrant [10], שמותאם לביצוע חיפושים מהירים במרחב רב-מימדי. המאגר מאורגן ב-"אוספים" (collections), כאשר כל אוסף מכיל מסמכים בנושא מסוים או ממקור מסוים. אינדקס זה נבנה פעם אחת, ואחר כך ניתן לחפש בו אינספור פעמים ללא צורך בעיבוד מחדש של המסמכים.

7.2.2 שלב השאילתה: הבנת כוונת המשתמש

כאשר משתמש שואל שאלה, אנחנו לא מחפשים התאמה מילולית במאגר. במקום זאת, אנחנו ממירים את השאילתה עצמה להטמעה ווקטורית באמצעות אותו מודל הטמעה שבו השתמשנו באינדקס. כך אנחנו מקבלים וקטור שמייצג את המשמעות הסמנטית של השאילתה.

למה זה חשוב? כי גישה זו מאפשרת לנו למצוא מסמכים רלוונטיים גם אם הם לא משתמשים באותן מילים בדיוק. אם המשתמש שואל "איך לשפר ביצועים של מודל?", המערכת תמצא גם מסמכים שמדברים על "אופטימיזציה של LLM", "שיפור יעילות מודלים",

ו-"טכניקות לשדרוג ביצועי בינה מלאכותית", למרות שאף אחד מהמסמכים הללו לא משתמש במילים המדויקות מהשאלתה. זהו היתרון המרכזי של חיפוש סמנטי על פני חיפוש מילות מפתח מסורתית.

7.2.3 שלב האחזור: מציאת המידע הרלוונטי

עכשיו מגיע השלב המרכזי: אנחנו משווים את וקטור השאלתה לכל הוקטורים במאגר ומחפשים את הקטעים הדומים ביותר. הדמיון נמדד באמצעות מדדים מתמטיים כמו דמיון קוסינוס (cosine similarity) או מרחק אוקלידי. דמיון קוסינוס מודד את הזווית בין שני וקטורים, כאשר ערך של 1.0 מציין זהות מוחלטת, ו-0.0 מציין חוסר קשר כלשהו. המאגר הווקטורי מחזיר את k הקטעים המובילים, כאשר k הוא מספר שאנחנו מגדירים מראש, בדרך כלל בין 3 ל-10. לכל קטע יש ציון דמיון, כך שאנחנו יודעים עד כמה הוא רלוונטי לשאלתה. לעתים אנחנו גם מגדירים סף מינימלי של דמיון, למשל 0.7, כדי לא להחזיר קטעים שהרלוונטיות שלהם נמוכה מדי. זה מונע מצב שבו המודל מקבל מידע לא רלוונטי שעלול להטעות אותו.

7.2.4 שלב הייצור: יצירת תשובה מושכלת

בשלב הסופי, אנחנו לוקחים את הקטעים שאחזרנו ומעבירים אותם למודל השפה הגדול יחד עם השאלתה המקורית. בעצם, אנחנו אומרים למודל: "הנה הקשר רלוונטי שמצאתי במאגר הידע, והנה השאלה. תן תשובה מבוססת על ההקשר הזה." המודל עכשיו יכול לענות על השאלה בצורה מדויקת יותר, כי יש לו גישה למידע ספציפי שמעבר לידע הכללי שלו. חשוב להבין שהמודל לא פשוט מעתיק מהמסמכים שאחזרנו, אלא מסנתז מהם מידע, משלב אותו עם הידע שלו, ויוצר תשובה קוהרנטית ומדויקת. זוהי העוצמה של Augmented Generation - המודל משתמש בידע החיצוני כבסיס, אך מוסיף את ההבנה העמוקה שלו בשפה והקשרים.

7.2.5 מודלי הטמעה: nomic-embed-text

בחירת מודל הטמעה היא החלטה קריטית בבניית מערכת RAG. מודל הטמעה הוא הכלי שממיר טקסט למספרים, ולכן איכותו משפיעה ישירות על יכולת המערכת למצוא מידע רלוונטי. בפרק זה נשתמש במודל nomic-embed-text [11], מודל פתוח ומתקדם שפותח על ידי חברת Nomic AI.

למה דווקא nomic-embed-text? המודל הזה משלב מספר יתרונות מרכזיים. ראשית, גודל חלון ההקשר - המודל תומך ב-8192 טוקנים, מה שמאפשר להטמיע מסמכים ארוכים במיוחד. שנית, רב-לשוניות - למרות שהמודל אומן בעיקר על אנגלית, הוא מראה ביצועים טובים גם בשפות אחרות, כולל עברית. שלישית, דיוק גבוה - במבחני MTEB (Massive Text Embedding Benchmark), המודל מציג ביצועים מצוינים ביחס לגודלו. ורביעית, קוד פתוח - המודל זמין בחינם ונותן לנו שליטה מלאה על תהליך ההטמעה.

הארכיטקטורה של nomic-embed-text מבוססת על BERT, אך כוללת שיפורים משמעותיים: מנגנון קשב משופר, טכניקות אימון מתקדמות, ויכולת להתמודד עם טקסט

ארוך במיוחד. המודל מחזיר וקטורים בגודל 768 מימדים, שהוא גודל סטנדרטי המאפשר חיפוש מהיר תוך שמירה על דיוק גבוה. למרות שזה נשמע כמספר גדול, זה למעשה פשרה מושכלת - מודלים קטנים יותר עם 384 מימדים מהירים יותר אך פחות מדויקים, בעוד מודלים גדולים יותר עם 1536 מימדים מדויקים יותר אך איטיים יותר ודורשים יותר זיכרון.

7.3 הגדרת vector_memory

כעת נעבור לצד המעשי: כיצד מגדירים מאגר וקטורי עבור סוכן? נתחיל בהתקנה והגדרה של Qdrant, מאגר וקטורי מתקדם שנבנה במיוחד לטיפול ביעילות בחיפושים סמנטיים במרחב רב-מימדי.

7.3.1 התקנת Qdrant באמצעות Docker

Qdrant הוא מאגר וקטורי בקוד פתוח שנכתב ב-Rust, מה שמבטיח ביצועים מצוינים ויעילות זיכרון גבוהה. הדרך הקלה והמהירה ביותר להריץ את Qdrant היא באמצעות Docker. כך נוכל להפעיל את המאגר בסביבה מבודדת, בלי להתקין תלויות נוספות במערכת. נריץ את הפקודה הבאה בטרמינל:

gnitsiL 11: הרצת Qdrant עם Docker

```
\ d- nur rekcod
\ revres-tnardq eman--
\ 3336:3336 p-
\ 4336:4336 p-
\ egarots/tnardq/:egarots_tnardq v-
tsetal:tnardq/tnardq
```

בואו נפרק את הפקודה הזו. הארגומנט `d- nur rekcod` מריץ קונטיינר חדש במצב רקע, כך שהוא ימשיך לפעול גם אחרי שנסגור את הטרמינל. הארגומנט `-tnardq eman--` נותן לקונטיינר שם מזהה, מה שמקל עלינו לנהל אותו מאוחר יותר. הארגומנט `3336:3336 p-` חושף את יציאת ה-REST API של Qdrant, שדרכה נתקשר עם המאגר. הארגומנט `4336:4336 p-` חושף את יציאת ה-gRPC לביצועים גבוהים יותר. הארגומנט `egarots/tnardq/:egarots_tnardq v-` יוצר נפח קבוע לשמירת הנתונים, כך שהם לא יאבדו כשנעצור את הקונטיינר. ולבסוף, `tsetal:tnardq/tnardq` הוא הדימוי הרשמי של Qdrant.

לאחר הרצת הפקודה, Qdrant זמין בכתובת `3336:tsohlaol//:ptth`. נוכל לבדוק שהשרת רץ כראוי על ידי ביקור בדפדפן בכתובת `draobhsad/3336:tsohlaol//:ptth`, שם נמצא ממשק ניהול גרפי נוח. הממשק מאפשר לנו לראות את כל האוספים, לבדוק סטטיסטיקות, ולבצע חיפושים ידניים למטרות ניפוי שגיאות.

7.3.2 יצירת Collection: הגדרת מבנה המאגר

ב-Qdrant, המידע מאורגן ב-collections, שהם בעצם טבלאות של וקטורים. כל collection מוגדר עם פרמטרים ספציפיים: גודל הוקטור, מדד המרחק, ואופן האינדקס. בואו ניצור collection חדש למסמכים שלנו.

הנה סקריפט Python שיוצר collection מתאים למודל nomic-embed-text:

gnitsiL 21: יצירת collection ב-Qdrant

```
tneilCtnardQ tropmi tneilc_tnardq morf
smaraProtceV , ecnatsiD tropmi sledom.tneilc_tnardq morf
```

```
tnardQ לשרת #
(3336 = trop , "tsohlaol" = tsoh) tneilCtnardQ = tneilc
```

```
שם האוסף #
"stnemucod_lacinhcet" = eman_noitcelloc
```

```
יצירת האוסף #
) noitcelloc_etaerc . tneilc
, eman_noitcelloc = eman_noitcelloc
) smaraProtceV = gifnoc_srotcev
txet-debme-cimon של # גודל וקטור
ENISOC.ecnatsiD = ecnatsiD # מדד דמיון קוסינוס
```

```
(
(
```

```
("yllufsseccus_etaerc" '{ eman_noitcelloc }' _noitcelloc "f) tnirp
```

בקוד לעיל, הגדרנו collection בשם stnemucod_lacinhcet עם וקטורים בגודל 768 מימדים. בחרנו במדד דמיון קוסינוס (Cosine Similarity), שהוא הבחירה הסטנדרטית למרבית יישומי RAG. מדד זה מודד את הזווית בין שני וקטורים, ולא את המרחק המוחלט ביניהם, מה שהופך אותו לחסין יותר לשינויים בגודל הווקטורים. זה חשוב במיוחד כי וקטורים שונים עשויים להיות בקנה מידה שונה, אך דמיון קוסינוס מתעלם מהקנה מידה ומתמקד רק בכיוון.

7.3.3 אינדקס מסמכים: הוספת תוכן למאגר

לאחר שיצרנו את ה-collection, הגיע הזמן להוסיף אליו מסמכים. התהליך כולל מספר שלבים: קריאת המסמכים, חלוקה לקטעים, יצירת הטמעות, והוספה למאגר. הנה דוגמה מלאה:

gnitsiL 31: אינדקס מסמכים ב-Qdrant

```

tneilCtnardQ tropmi tneilc_tnardq morf
tcurtStnioP tropmi sledom.tneilc_tnardq morf
remrofsnarTecnetneS tropmi sremrofsnart_ecnetnes morf
diuu tropmi

```

טעינת מודל ההטמעה

```

(' 1v- txet-debme-cimon/ia-cimon') remrofsnarTecnetneS = ledom

```

התחברות למאגר

```

(3336= trop , "tsohlaol"=tsoh) tneilCtnardQ = tneilc
"stnemucod_lacinhcet" = eman_noitcelloc

```

מסמכי דוגמה

```

] = stnemucod
}

```

```

"laveirter_senibmoc_taht_euqinhcet_a_si_GAR" : "txet"
, ".sesnopser_MLL_devorpmi_rof_noitareneg_htiw"
{1 : "egap" , "7retpahc" : "ecruos"} : "atadatem"

```

```

,{
}

```

```

"elbane_dna_sgniddebme_erots_sesabatad_rotceV" : "txet"
, ".hcraes_ytiralis_citnames"
{2 : "egap" , "7retpahc" : "ecruos"} : "atadatem"

```

```

,{
}

```

```

"gniddebme_ecruos-nepo_na_si_txet-debme-cimoN" : "txet"
, ".wodniw_txetnoc_nekot_2918_htiw_ledom"
{3 : "egap" , "7retpahc" : "ecruos"} : "atadatem"

```

```

{

```

```

[

```

יצירת נקודות (stniop) עבור tnardQ

```

[] = stniop

```

```

:stnemucod ni cod rof

```

יצירת gniddebme

```

() tsilot.([ "txet"]cod)edocne.ledom = gniddebme

```

יצירת tniop עם DI ייחודי

```

) tcurtStnioP = tniop

```

```

, (( 4diuu . diuu ) rts = di
, gniddebme = rotcev
} = daolyap
, [ "txet" ] cod : "txet"
[ "atadatem" ] cod **
{
(
( tniop ) dneppa . stniop

# הוספה למאגר
) trespu . tneilc
, eman_noitcelloc = eman_noitcelloc
stniop = stniop
(

```

```

( " yllufsseccus _ stnemucod _ { ( stniop ) nel } _ daxednI " f ) tnirp

```

הקוד לעיל מדגים את התהליך המלא. אנחנו טוענים את מודל ההטמעה, עוברים על כל מסמך, ממירים אותו לווקטור, ומוסיפים אותו למאגר יחד עם המטא-דאטה. המטא-דאטה (payload) מכילה את הטקסט המקורי ומידע נוסף כמו מקור המסמך ומספר העמוד, שיהיה שימושי כשנרצה להציג למשתמש מהיכן הגיעה התשובה. שימוש ב-UUID כמזהה מבטיח שכל נקודה תקבל מזהה ייחודי, גם אם נריץ את הסקריפט מספר פעמים.

7.3.4 תצורת YAML מלאה: חיבור הסוכן למאגר

כעת נגדיר את החיבור בין הסוכן למאגר הווקטורי בקובץ התצורה YAML. התצורה כוללת שני רכיבים מרכזיים: הגדרת ה-`vector_memory` עצמו, והגדרת מודל ההטמעה.

gnitsiL 41: תצורה מלאה של `vector_memory`

```

: tneg a
"tnega-egdelwonk-maroy" : eman
"rosivda_lacinhcet" : elor

# הגדרת מודל ההטמעה
: sgniddebme
"amallo" : redivorp -
"txet-debme-cimon" : ledom
867 : snoisnemid
2918 : wodniw_txetnoc
"43411:tsohlacol//:ptth" : esab_ipa

```

```

# הגדרת זיכרון וקטורי
: yromem_rotcev
- "tnardq" : redivorp
"stnemucod_lacinhcet" : eman_noitcelloc
"tsohlaol" : tsoh
3336 : trop
"txet-debme-cimon" : ledom_gniddebme
5 : k_pot
7.0 : dlohserht_erocs
"enisoc" : cirtem_ecnatsid

# פרומפט מערכת מעודכן
| : tpmorp_metsys
.stnega IA ni gnizilaiceps rosivda lacinhcet a , maroY era uoY
if esab egdelwonk ruoy hcraes syawla , snoitseuq gnirewsna nehW
.elbaliava nehW txetnoc deveirter eht no sesnopser ruoy esaB
ruoy_esu_,noitamrofni_tnaveler_niatnoc_t'nseod txetnoc eht fl
.noitatimil_siht_noitnem_tub_egdelwonk_lareneg_
.noitamrofni_deveirter_gnisu_nehw_secruos_ruoy_etic_syawla_

```

בתצורה זו הגדרנו מספר פרמטרים חשובים. הפרמטר `k_pot` : 5 מורה למערכת להחזיר את חמשת הקטעים הרלוונטיים ביותר. הפרמטר `dlohserht_erocs` : 7.0 מבטיח שנחזיר רק קטעים עם ציון דמיון של לפחות 0.7, מה שמונע החזרת מידע לא רלוונטי. הפרמטר `cirtem_ecnatsid` : "enisoc" מציין שימוש במדד דמיון קוסינוס. והפרמטר `ledom_gniddebme` מפנה למודל ההטמעה שהגדרנו בסעיף `sgniddebme`. שימו לב גם לשינוי בפרומפט המערכת: הוספנו הוראות ברורות לסוכן לחפש במאגר הידע ולצטט את המקורות. זה מבטיח שהסוכן ינצל את היכולת החדשה שקיבל. הפרומפט גם מנחה את הסוכן מה לעשות כשהמאגר לא מכיל מידע רלוונטי - להשתמש בידע הכללי שלו, אך לציין זאת במפורש. שקיפות זו חשובה למשתמשים כדי שידעו על מה הם יכולים לסמוך.

7.4 הגדרת knowledge Retriever – פרוטוקול MCP

בעוד שהסעיף הקודם התמקד בהגדרת המאגר הווקטורי עצמו, כעת נעבור לשאלה קריטית: כיצד הסוכן שלנו מתקשר עם המאגר? האם הוא שולח שאילתות HTTP ישירות? האם הוא קורא למודול Python? התשובה היא שום דבר מהנ"ל. במקום זאת, נשתמש בפרוטוקול סטנדרטי שפותח במיוחד למטרה זו: Model Context Protocol (MCP) [12].

7.4.1 מהו Model Context Protocol?

MCP הוא פרוטוקול תקשורת שפותח על ידי Anthropic במטרה ליצור ממשק אחיד בין

מודלי שפה לבין מקורות מידע חיצוניים. הרעיון דומה לאופן שבו HTTP מאפשר לדפדפנים שונים לתקשר עם שרתי אינטרנט שונים, גם אם הם נכתבו בטכנולוגיות שונות לחלוטין. לפני MCP, כל מערכת RAG הייתה צריכה לממש את החיבור למאגר הווקטורי בצורה ייחודית. אם רצית לעבור מ-Qdrant ל-Pinecone, היית צריך לשכתב קוד משמעותי. אם רצית להוסיף מקור מידע נוסף, כמו מסד נתונים רלציוני או ממשק API חיצוני, שוב היית צריך לפתח אינטגרציה ייחודית. MCP פותר את הבעיה הזו על ידי הגדרת ממשק סטנדרטי שכל מאגר וקטורי יכול ליישם.

הפרוטוקול מגדיר מספר פעולות בסיסיות. פעולת `hcraes` מבצעת חיפוש סמנטי במאגר הידע. פעולת `tresni` מוסיפה מסמכים חדשים למאגר. פעולת `etadpu` מעדכנת מסמכים קיימים. פעולת `eteled` מוחקת מסמכים מהמאגר. ופעולת `snoitcelloc_tsil` מחזירה רשימת אוספים זמינים.

בנוסף, MCP מגדיר מבנה תקשורת ברור: הסוכן שולח בקשה בפורמט JSON, השרת מעבד אותה, ומחזיר תשובה גם היא בפורמט JSON. הפשטות הזו מאפשרת אינטגרציה קלה עם כל סוג של מאגר או מקור מידע. זו הרבה יותר מסתם תקן טכני - זו אבסטרקציה שמפרידה בין הלוגיקה של הסוכן לבין הפרטים הטכניים של אחסון המידע.

7.4.2 מבנה שרת MCP לאחזור ידע

שרת MCP הוא תוכנית קטנה שפועלת כמתווך בין הסוכן למאגר הווקטורי. הוא מאזין לבקשות מהסוכן, מתרגם אותן לפעולות על המאגר, ומחזיר את התוצאות בפורמט מתאים. בואו נבנה שרת MCP פשוט עבור Qdrant:

gnitsiL 51: שרת MCP לאחזור ידע מ-Qdrant

```
loot , revreSPCM tropmi revres.pcm morf
tneilCtnardQ tropmi tneilc_tnardq morf
remrofsnarTecnetneS tropmi sremrofsnart_ecnetnes morf
yna , tciD , tsil tropmi gnipytt morf
nosj tropmi
```

```
: ( revreSPCM ) reveirteRegdelwonK ssalc
, tni : trop_tnardq , rts : tsoh_tnardq , fles ) __tini__ fed
: ( rts : ledom_gniddebme , rts : eman_noitcelloc
( " reveirter - egdelwonk " = eman ) __tini__ . ( ) repus
```

אתחול לקוחות

```
) tneilCtnardQ = tneilc.fles
, tsoh_tnardq = tsoh
trop_tnardq = trop
```

(

```
( ledom_gniddebme ) remrofsnarTecnetneS = ledom.fles
```



```
eman_noitcelloc = eman_noitcelloc.fles
```

```
# רישום הכלים
```

```
() sloot_retsiger.fles
```

```
:( fles ) sloot_retsiger fed
```

```
" " "רישום הכלים הזמינים לסוכן" " "
```

```
) loot.fles@
```

```
, "egdelwonk_hcraes" = eman
```

```
"_tnaveler_rof_esab_egdelwonk_eht_hcraes" = noitpircsed
```

```
"ytiralimis_citnames_gnisu_noitamrofni"
```

```
(
```

```
) egdelwonk_hcraes fed cnysa
```

```
, rts : yreuq
```

```
, 5 = tni : k_pot
```

```
7.0 = taolf : dlohserht_erocs
```

```
: [[ ynA , rts ] tciD ] tsiL <- (
```

```
" " "חיפוש סמנטי במאגר הידע" " "
```

```
# יצירת gniddebme לשאלתה
```

```
() tsilot.( yreuq ) edocne.ledom.fles = rotcev_yreuq
```

```
# חיפוש ב-tnardQ
```

```
) hcraes.tneilc.fles = stluser
```

```
, eman_noitcelloc.fles = eman_noitcelloc
```

```
, rotcev_yreuq = rotcev_yreuq
```

```
, k_pot = timil
```

```
dlohserht_erocs = dlohserht_erocs
```

```
(
```

```
# פורמט התוצאות
```

```
[] = stluser_dettamrof
```

```
: stluser ni tluser rof
```

```
) dneppa.stluser_dettamrof
```

```
, ( " " , "txet" ) teg.daolyap.tluser : "txet"
```

```
, erocs.tluser : "erocs"
```

```
} : "atadatem"
```

```
() smeti.daolyap.tluser ni v , k rof v : k
```

```

"txet" =! k fi
{
({

stluser_dettamrof nruter

)loot.fles@
,"tnemucod_tresni"=eman
"eh_otni_tnemucod_wen_a_tresni"=noitpircsed
"esab_egdelwonk"

(
)tnemucod_tresni fed cnysa
, rts :txet
enoN = [ynA , rts] tciD :atadatem
:[ rts , rts] tciD <- (
"" "הוספת מסמך חדש למאגר" ""

diuu tropmi
tcurtStnioP tropmi sledom.tneilc_tnardq morf

gniddebme # יצירת #
() tsilot.(txet)edocne.ledom.fles = gniddebme

tniop # יצירת #
)tcurtStnioP = tniop
, (()) 4diuu.diuu) rts=di
, gniddebme=rotcev
}=daolyap
, txet : "txet"
({} ro atadatem)**
{
(

# הוספה למאגר
)trespu.tneilc.fles
, eman_noitcelloc.fles=eman_noitcelloc
[tniop]=stniop
(

```

```

    } nruter
    , "sseccus" : "sutats"
    '{di.tniop}_DI_htiw_detresni_tnemucoD"f : "egassem"
    {

    : "__niam__" == __eman__ fi
    # יצירת והרצת השרת
    )reveirteRegdelwonK = revres
    , "tsohlaol" = tsoh_tnardq
    , 3336 = trop_tnardq
    , "stnemucod_lacinhcet" = eman_noitcelloc
    "1v- txet- debme- cimon/ia- cimon" = ledom_gniddebme

    (

    (0008 = trop) nur. revres

```

שרת ה-MCP הזה מספק שני כלים עיקריים: `egdelwonk_hcraes` לחיפוש מידע, ו-`tnemucod_tresni` להוספת מסמכים חדשים. כל כלי מוגדר עם שם, תיאור, ופרמטרים ברורים. הסוכן יכול לקרוא לכלים הללו בדיוק כמו שהוא קורא לכלים אחרים שהכרנו בפרקים קודמים. החידוש כאן הוא שהכלים האלה נותנים לסוכן גישה למאגר ידע עצום, מה שמרחיב משמעותית את היכולות שלו.

7.4.3 אינטגרציה עם הסוכן: תצורת MCP

כעת נחבר את שרת ה-MCP לסוכן שלנו. נוסיף את ההגדרה הבאה לקובץ ה-YAML של הסוכן:

gnitsiL 61: אינטגרציה של שרת MCP בתצורת הסוכן

```

: tneg a
"tnega-egdelwonk-maroy" : eman

# הגדרת שרתי PCM
: srevres_pcm
: egdelwonk
"nohtyp" : dnammoc
: sgra
"yp.reveirter_egdelwonk/ot/htap/:C" -
: vne
"tsohlaol" : TSOH_TNARDQ
"3336" : TROP_TNARDQ
"stnemucod_lacinhcet" : EMAN_NOITCELLOC

```

"lv-txet-debme-cimon/ia-cimon" : LEDOM_GNIDDEBME

כלים זמינים נוספים (אוטומטית מ-PCM)

:sloot

egdelwonk_hcraes -

tnemucod_tresni -

| :tpmorp_metsys

tsav a ot ssecca htiw rosivda lacinhcet a ,maroY era uoY
.esab egdelwonk

:snoitseuq gnirewsna nehW

tsrif loot egdelwonk_hcraes eht esu SYAWLA .1

ylluferac txetnoc deveirter eht weiver .2

tnaveler nehW txetnoc eht no rewsna ruoy esaB .3

dedivorp atadatem eht gnisu secruos etiC .4

ylraelc siht noitnem , tneiciffusni si txetnoc fl .5

nehW tnmucod_tresni gnisu egdelwonk wen dda osla nac uoY
.snoitasrevnoc gnirud wen gnihtemos nrael uoy

התצורה הזו מגדירה כיצד להפעיל את שרת ה-MCP: באיזה פקודה, עם אילו ארגומנטים, ועם אילו משתני סביבה. כאשר הסוכן מופעל, הוא אוטומטית מתחיל את שרת ה-MCP, מגלה את הכלים הזמינים, ומוסיף אותם לארסנל שלו. זה קורה באופן שקוף לחלוטין - המשתמש לא צריך לדאוג להתקנה או הגדרה נוספת.

7.5 הוספת RAG לסוכן Yoram-Agent-Knowledge

כעת נראה את התמונה המלאה: כיצד לשדרג סוכן קיים להיות סוכן עם יכולות RAG מלאות. נלקח את Yoram, הסוכן שפיתחנו בפרק שישי, ונוסיף לו גישה למאגר ידע עצום.

7.5.1 תצורת Yoram ללא RAG (מפרק שישי)

הנה התצורה המקורית של Yoram מפרק שישי, בטרם הוספנו יכולות RAG:

gnitsiL 71: תצורת Yoram המקורית (ללא RAG)

:tnega

"rosivda-cimedaca-maroy" :eman

"trep_xe_noitatic_cimedaca" :elor

:ledom

```

"ciporhtna" : redivorp
"5-4-tennos-edualc" : eman
3.0 : erutarepmet
0008 : snekot_xam

: sloot
"bew_hcraes" : eman -
"bew_eht_no_secruos_cimedaca_hcraes" : noitpircsed
"noitatic_tamrof" : eman -
"elyts_EEEI_ni_snoitatic_tamroF" : noitpircsed
"ecnerefer_etadilav" : eman -
"ssenetelpmoc_ecnerefer_etadilaV" : noitpircsed

| : tpmorp_metsys
.tsilaiceps noitatic cimedaca na , maroY era uoY
secnerefer cimedaca tamrof dna dnif srehcraeser pleh uoY
.sdradnats EEEI ot gnidrocca

```

```

: syawlA
secruos evitatiohtua rof hcraes -
sliated noitacilbup yfireV -
yltnetsisnoc snoitatic tamroF -
noitamrofni cihpargoilbib etelpmoc edivorp -

```

זוהי תצורה טובה, אך Yoram מוגבל לידע הכללי שלו ולמה שהוא יכול למצוא בחיפוש רשת מיידי. אם אנחנו רוצים שהוא יוכל לגשת לספריה שלמה של מאמרים אקדמיים ששמרנו מראש, או למאגר פנימי של מחקרים שעשינו, נצטרך להוסיף RAG.

7.5.2 תצורת Yoram עם RAG מלא

הנה התצורה המשודרגת של Yoram, עם יכולות RAG מלאות:

gnitsiL 81: תצורת Yoram המשודרגת (עם RAG)

```

: tnegA
"rosivda-cimedaca-maroy" : eman
"trep_xe_noitatic_cimedaca" : elor

: ledom
"ciporhtna" : redivorp
"5-4-tennos-edualc" : eman
3.0 : erutarepmet

```

```

0008 : snekot_xam

# הוספת מודל הטמעה
: sgniddebme
"amallo" : redivorp -
"txet-debme-cimon" : ledom
867 : snoisnemid
2918 : wodniw_txetnoc
"43411:tsohlaacol//:ptth" : esab_ipa

# הוספת זיכרון וקטורי
: yromem_rotcev
"tnardq" : redivorp -
"srepap_cimedaca" : eman_noitcelloc
"tsohlaacol" : tsoh
3336 : trop
"txet-debme-cimon" : ledom_gniddebme
7 : k_pot
57.0 : dlohserht_erochs
"enisoc" : cirtem_ecnatsid
: retlif_atadatem
"repap_cimedaca" : epyt
eurt : deifirev

# הוספת שרת PCM לאחזור ידע
: srevres_pcm
: egdelwonk
"nohtyp" : dnammoc
: sgra
"yp.reveirter_egdelwonk/maroy/stnega/:C" -
: vne
"tsohlaacol" : TSOH_TNARDQ
"3336" : TROP_TNARDQ
"srepap_cimedaca" : EMAN_NOITCELLOC
"lv-txet-debme-cimon/ia-cimon" : LEDOM_GNIDDEBME

# כלים משודרגים
: sloot
"esab_egdelwonk_hcraes" : eman -

```

```

"esab_egdelwonk_repap_cimedaca_egt_hcraes" :noitpircsed
    "egdelwonk" :revres_pcm
    "egdelwonk_hcraes" :loot_pcm
    "bk_ot_repap_dda" :eman -
"esab_egdelwonk_ot_repap_cimedaca_wen_a_dddA" :noitpircsed
    "egdelwonk" :revres_pcm
    "tnemucod_tresni" :loot_pcm
    "bew_hcraes" :eman -
    "bew_egt_no_secruos_cimedaca_hcraes" :noitpircsed
    "noitatic_tamrof" :eman -
    "elyts_EEEI_ni_snoitatic_tamroF" :noitpircsed
    "ecnerefer_etadilav" :eman -
    "ssenetelpmoc_ecnerefer_etadilaV" :noitpircsed

```

```

| :tpmorp_metsys
ssecca htiw tsilaiceps noitatic cimedaca na ,maroY era uoY
    .srepap cimedaca fo esab egdelwonk evisnetxe na ot

```

```

:WOLFKROW
esab egdelwonk ruoy hcraes TSRIFF ,cipot a tuoba dekxa nehW .1
    esab_egdelwonk_hcraes gnisu
    ylluferac srepap deveirter egt weiver .2
    egt sa ti esu ,noitamrofni tnaveler sah esab egdelwonk fl .3
    ecruos yramirp
    dnif ot bew_hcraes esu ,tneiciffusni si esab egdelwonk fl .4
    secruos lanoitidda
    egt ot ti dda ,repap wen elbaulav a dnif uoy nehW .5
    bk_ot_repap_dda gnisu esab egdelwonk

```

```

:SELUR NOITATIC
    snoitatic dettamrof-EEEI edivorp syawlA -
    elbaliava nehW IOD edulcni -
    gnicnerefer nehW egap/noitces cificeps egt etiC -
    gnitic erofeb sliated noitacilbup yfireV -

```

```

:EGASU ESAB EGDELWONK
    ,tcartsba ,srohtua ,eltit :sedulcni txetnoc deveirter -
    IOD ,raey noitacilbup
    ecnaveler ssessa ot erocs.atadatem egt esU -

```

```
noitatic tcerid rof 8.0 < erocs htiw srepap referP -
dnuorgkcab tnaveler era 8.0-57.0 erocs htiw srepap -
tnaveler yltcerid eb ton yam 57.0 > erocs htiw srepap -
```

שימו לב לשינויים המרכזיים בתצורה המשודרגת. ראשית, הוספנו הגדרה של מודל ההטמעה nomic-embed-text דרך Ollama. שנית, הגדרנו חיבור ל-Qdrant עם collection ייעודי למאמרים אקדמיים. שלישית, הוספנו פילטור מטא-דאטה שמוודא שרק מאמרים מסוג repap_cimedaca שמסומנים כ-deifirev יוחזרו. רביעית, הוספנו שני כלים חדשים - esab_egdelwonk_hcraes ו-bk_ot_repap_dda. וחמישית, הרחבנו את הפרומפט המערכת עם הנחיות מפורטות על שימוש במאגר הידע.

7.5.3 פרומפט מערכת מתקדם: שילוב RAG בתהליך החשיבה

הפרומפט המערכת המשודרג ב-Yoram הוא קריטי להצלחת השילוב. הוא לא רק מספר לסוכן שיש לו גישה למאגר ידע, אלא גם מנחה אותו כיצד להשתמש בו בצורה אפקטיבית. בואו נפרק את המרכיבים השונים.

תהליך עבודה מובנה - הפרומפט מגדיר רצף פעולות ברור. קודם כל, חיפוש במאגר. אחר כך, הערכת התוצאות. במידת הצורך, חיפוש חיצוני נוסף. ולבסוף, הוספת ידע חדש למאגר. זה מבטיח ש-Yoram ינצל את מאגר הידע באופן עקבי ושיטתי.

כללי ציטוט - הפרומפט מדגיש את חשיבות הציטוט המדויק ושימוש ב-DOI. כאשר Yoram מאחזר מאמר מהמאגר, הוא יודע בדיוק כיצד לצטט אותו. הוא גם יודע לכלול את הסעיף או העמוד הספציפי שממנו נלקח המידע, לא רק את המאמר כולו.

פרשנות ציוני דמיון - הפרומפט מלמד את Yoram כיצד לפרש את ציוני הדמיון של התוצאות. מאמרים עם ציון מעל 0.8 רלוונטיים מאוד ומתאימים לציטוט ישיר. מאמרים עם ציון בין 0.75 ל-0.8 רלוונטיים כרקע כללי. ומאמרים עם ציון נמוך מ-0.75 עשויים להיות רעש ולא כדאי להסתמך עליהם. זה מונע מהסוכן להסתמך על מידע לא רלוונטי ומבטיח איכות גבוהה של התשובות.

7.5.4 דוגמה מעשית: Yoram בפעולה עם RAG

כעת נראה כיצד Yoram המשודרג עובד בפועל. נניח שחוקר פונה אלינו עם השאלה הבאה: "I need recent papers on transformer architectures for multilingual NLP, specifically works that address right-to-left languages like Hebrew and Arabic."

הנה כיצד Yoram מגיב, שלב אחר שלב. ראשית, הוא חושב: "User is asking for specific academic papers. I should search my knowledge base first. קריאה לכלי esab_egdelwonk_hcraes עם השאילתה "transformer architectures multilin-". ופרמטרים k_pot=7 ו-dlohserht_erocs=57.0.

מאגר הידע מחזיר חמישה מאמרים. המאמר הראשון הוא "mBERT and Multilingual Transformers: Challenges in RTL Languages" מאת Cohen, Levy, and Globerson משנת 2023,

עם ציון 0.89. המאמר השני הוא "Bidirectional Attention in Hebrew BERT Models" מאת Tsarfaty, Seker, and Elazar משנת 2022, עם ציון 0.85. יש עוד שלושה מאמרים נוספים. Yoram חושב: "Excellent! I found highly relevant papers with scores > 0.8. I'll base my response on these." ואז הוא מגיב למשתמש עם רשימה מפורטת של הציטוטים בפורמט IEEE, כולל הסברים קצרים על כל מאמר. הוא מציין שהמאמר הראשון מתייחס ישירות לארכיטקטורות טרנספורמר עבור שפות RTL, ובוחן גם עברית וגם ערבית. המאמר השני מציג את HeBERT, המדגים שיפורים משמעותיים בהבנת השפה העברית דרך מנגנוני קשב משופרים.

בסוף התשובה, Yoram שואל: "Would you like me to search for additional papers in specific sub-areas, or would you like more details about any of these works?" אינטראקטיבית ומועילה.

שימו לב לאופן שבו Yoram עבד. הוא מיד חיפש במאגר הידע, לא הסתמך רק על הידע הכללי שלו. הוא קיבל תוצאות עם ציוני דמיון גבוהים, מה שאישר רלוונטיות. הוא פירמט את הציטוטים בסגנון IEEE מדויק. הוא הוסיף הערות קצרות על כל מאמר, מבוססות על התקציר. והוא הציע צעדים הבאים, שומר על השיחה אינטראקטיבית.

עכשיו נניח שהמשתמש מבקש מידע על מאמר ש-Yoram לא מצא במאגר: "Do you have anything on the recent GPT-4 multilingual benchmarks published this month?"

Yoram קודם חושב: "User asking about very recent work. Let me search knowledge base first." הוא מבצע חיפוש במאגר הידע עם השאילתה "GPT-4 multilingual benchmarks recent work. I should search the web and potentially add it to the knowledge base." אז הוא חושב: "Knowledge base doesn't have this". באינטרנט עם bew_hcraes ומוצא מאמר חדש.

לאחר מכן, Yoram מוסיף את המאמר למאגר הידע באמצעות bk_ot_repap_dda, כולל כל המטא-דאטה הרלוונטית: כותרת, מחברים, שנה, סוג, סטטוס אימות, ו-DOI. לבסוף, הוא מגיב למשתמש עם הציטוט המעוצב, כולל הערה: "Note: I've added this paper to my knowledge base for future reference."

הדוגמה הזו מדגימה את המעגל המלא של RAG. Yoram קודם חיפש במאגר הידע שלו, לא מצא, פנה לחיפוש חיצוני, ולבסוף הוסיף את המידע החדש למאגר. בפעם הבאה שמישהו ישאל על אותו מאמר, Yoram ימצא אותו מיידית במאגר הפנימי שלו. זהו כוחה של ארכיטקטורת RAG - היא לא רק מאפשרת גישה לידע קיים, אלא גם לומדת ומתעדכנת כל הזמן, והופכת לחכמה יותר עם כל אינטראקציה.

7.6 סיכום ומבט קדימה

בפרק זה חקרנו את עולם ה-RAG והזיכרון הווקטורי, טכנולוגיות שמשנות באופן יסודי את יכולות הסוכנים האוטונומיים שלנו. ראינו כיצד מגבלת חלון ההקשר, שנראתה בלתי ניתנת להתגברות, נפתרת באלגנטיות באמצעות שילוב של הטמעות סמנטיות, מאגרי וקטורים, ופרוטוקולי תקשורת סטנדרטיים.

למדנו את העקרונות המרכזיים של Retrieval-Augmented Generation: שלב האינדקס

שבו אנחנו מפרקים מסמכים לקטעים וממירים אותם לוקטורים, שלב השאילתה שבו אנחנו ממירים את שאלת המשתמש לוקטור, שלב האחזור שבו אנחנו מוצאים את הקטעים הדומים ביותר, ושלב הייצור שבו אנחנו יוצרים תשובה מבוססת על הידע שאחזרנו. הבנו כיצד מודלי הטמעה כמו nomic-embed-text ממירים טקסט למרחב ווקטורי שבו משמעות סמנטית מתורגמת למרחק מתמטי.

ראינו כיצד Qdrant מספק פלטפורמה יעילה לאחסון וחיפוש של מיליוני וקטורים. למדנו כיצד ליצור collections, להוסיף מסמכים, ולבצע חיפושים סמנטיים. חקרנו את Model Context Protocol, סטנדרט חדש שמפשט את האינטגרציה בין סוכנים למקורות ידע חיצוניים. בנינו שרת MCP מלא לאחזור ידע, וראינו כיצד הוא משתלב בצורה חלקה בתצורת הסוכן.

לבסוף, שדרגנו את Yoram, הסוכן האקדמי שלנו, ממומחה ציטוטים רגיל לסוכן עם גישה למאגר ידע עצום. ראינו בדוגמאות מעשיות כיצד הוא משתמש במאגר, מעריך רלוונטיות, ואף מוסיף ידע חדש באופן דינמי. התוצאה היא סוכן שלא רק חכם, אלא גם בעל זיכרון ארוך טווח ויכולת למידה מתמשכת.

אך זוהי רק ההתחלה. בפרקים הבאים נראה כיצד לייעל את מערכות ה-RAG שלנו. נלמד על שיטות מתקדמות לחלוקת מסמכים, כמו recursive chunking שמתחשב במבנה המסמך, ו-semantic chunking שמפצל לפי משמעות ולא רק לפי גודל. נחקור אסטרטגיות לדירוג מחדש של תוצאות, כמו cross-encoder reranking שמשפר את הדיוק. ונלמד טכניקות לטיפול בשאילתות מורכבות הדורשות מידע ממקורות מרובים, כמו multi-hop reasoning. נחקור גם את האתגרים הייחודיים של RAG רב-לשוני, במיוחד עבור שפות כמו עברית שדורשות טיפול מיוחד. איך מטפלים בקידומות וסיומות בעברית? איך מבטחים שהחיפוש הסמנטי עובד טוב גם בשפות RTL? ואיך משלבים מסמכים בשפות שונות באותו מאגר? הדרך ארוכה, אך כעת יש לנו את הכלים הבסיסיים לבנות סוכנים שלא רק חכמים, אלא גם בעלי זיכרון עמוק ויכולת גישה לידע כמעט בלתי מוגבל. זהו יסוד מהותי לכל סוכן ארגוני אמיתי שצריך לפעול במציאות מורכבת עם כמויות עצומות של מידע.

7.7 הסבר עקרוני: סיכונים ובקרה על פקודות מערכת

כאשר אנו מעניקים לסוכן AI את היכולת להריץ פקודות Shell ישירות על המערכת המארחת, אנו למעשה פותחים שער אל עולם של אפשרויות אדירות – אך גם של סיכונים משמעותיים. כלי ה-bash הוא אולי הכלי החזק והגמיש ביותר שניתן להעניק לסוכן, שכן הוא מאפשר גישה כמעט בלתי מוגבלת למערכת הקבצים, לתהליכים, לרשת ולמשאבים אחרים. אך בדיוק מסיבה זו, הוא גם הכלי המסוכן ביותר.

בפרק זה, נחקור את הממד האבטחתי של כלים מערכתיים בסביבה Agentic, ונלמד כיצד ליישם את עיקרון ההרשאה המינימלית (Principle of Least Privilege) כדי לאפשר לסוכן לבצע משימות דיאגנוסטיות הכרחיות, תוך מניעת נזקים פוטנציאליים למערכת. נמשיך בפיתוח סוכן ה-expert-analyst שלנו, "יורם", ונוסיף לו יכולות bash מבוקרות ומאובטחות.

7.7.1 העוצמה והסכנה של כלי ה-Bash

כלי ה-bash מאפשר לסוכן להריץ כל פקודת Shell שהמשתמש המארח יכול להריץ. זה כולל, בין היתר:

- קריאה, כתיבה ומחיקה של קבצים וספריות

- הרצת תוכניות וסקריפטים חיצוניים

- שינוי הרשאות קבצים ותצורת מערכת

- גישה לרשת ולמשאבים מרוחקים

- ניהול תהליכים ושירותים

בסביבת Production או כאשר הסוכן מופעל עם הרשאות מנהל (Administrator/root), השימוש הלא זהיר בכלי זה עלול לגרום לנזקים בלתי הפיכים: מחיקת נתונים קריטיים, חשיפת מידע רגיש, שיבוש שירותים פעילים, או אפילו פגיעה באבטחת המערכת כולה [13]. נבחן מקרה דוגמה: נניח שהמשתמש מבקש מהסוכן "נקה קבצים זמניים ישנים". סוכן שאינו מוגבל עשוי לפענח את הבקשה כ-`rm -rf /tmp/*` – פקודה שעשויה למחוק גם קבצים זמניים קריטיים של תהליכים פעילים. אם הסוכן מופעל עם הרשאות root, הנזק עלול להתפשט אף מעבר לתיקיית `/tmp`. מקרה חמור עוד יותר: בקשה לא מדויקת כמו "מצא ומחק קבצים כפולים" עלולה להוביל לפקודה מסוג:

פקודה מסוכנת - דוגמה למניעה

```
find / -type f -exec md5sum {} \; | sort | uniq -d -w32 | xargs rm
```

פקודה כזו, ללא אמצעי בקרה, עלולה למחוק קבצים קריטיים המערכת מזהה כ"כפולים" אך למעשה נדרשים לתפקוד תקין.

7.7.2 עיקרון ההרשאה המינימלית: Principle of Least Privilege

עיקרון ההרשאה המינימלית הוא אבן יסוד באבטחת מידע [14]. העיקרון קובע כי כל ישות (משתמש, תוכנית, או במקרה שלנו – סוכן AI) צריכה לקבל רק את ההרשאות המינימליות הנדרשות לביצוע המשימות שלה – ולא יותר. עבור סוכן ה-expert-analyst שלנו, המטרה היא ניתוח ושליפת מידע. בהקשר זה, יכולות ה-bash נדרשות למטרות מוגבלות בלבד:

- אבחון מצב המערכת (דוגמה: `df -h, free -m`)

- קבלת רשימות קבצים או מידע על סביבת העבודה (דוגמה: `ls -la, env`)

- קריאת תוכן קבצים בדרכים שכלי `read` אינו תומך (נדיר)

אין להם צורך לשנות, למחוק או לכתוב קבצים. אין להם צורך להריץ סקריפטים חיצוניים לא מאומתים. ובוודאי אין להם צורך בגישה לפקודות מנהליות מסוכנות. לכן, הפתרון המתבקש הוא: להעניק לסוכן גישה מוגבלת וביקורתית לכלי `bash`, תוך הטמעת מנגנוני בטיחות בתוך ההגדרות עצמן.

7.7.3 bashoutput לעומת bash: שתי רמות בקרה

במסגרת Claude CLI [15], קיימים שני כלים קשורים המאפשרים ביצוע פקודות מערכת:

1. **bash** – מאפשר הרצת פקודות Shell בודדות ומחזיר את הפלט. כלי זה נותן גמישות רבה, אך דורש ניהול זהיר.

2. **bashoutput** – מתמקד בהרצת פקודה והחזרת הפלט בלבד, עם שכבת אבטחה מוגברת יחסית. כלי זה מועדף למצבים בהם אנו מעוניינים רק בקריאת מידע, ולא בשינוי מצב המערכת.

האסטרטגיה המומלצת עבור סוכן כמו expert-analyst:

- העדפה ל-bashoutput לכל המקרים בהם רק קריאה נדרשת.

- שימוש מוגבל ב-bash רק כאשר נדרשת גמישות נוספת, ותמיד בליווי הנחיות אבטחה מחמירות ב-system_prompt.

- אי-הכללת כלים מסוכנים כמו `write, edit, notebookedit` – כלים אלו מאפשרים שינוי ישיר של קבצים ללא ביקורת, והם מהווים וקטור תקיפה פוטנציאלי.

7.7.4 תרחישי סיכון וצעדי מניעה

נבחן מספר תרחישי סיכון אופייניים ואת הפתרונות שלהם. טבלה 3 מציגה ארבעה תרחישים נפוצים והפתרונות המומלצים להם:

טבלה 3: תרחישי סיכון וצעדי הגנה במערכת Agentic

תרחיש	התרחיש המסוכן	צעד מניעה
1. מחיקה בשוגג	הסוכן מריץ <code>rm, mv, chmod</code> ללא כוונה זדונית, פשוט בתגובה לפרשנות מוטעית של בקשת משתמש	הנחיית הוספת אבטחה מפורשת ב- <code>system_prompt</code> : "אסור להריץ פקודות המשנות קבצים"
2. שינוי תצורה ללא ביקורת	משתמש מבקש "עדכן קובץ תצורה", והסוכן עושה זאת ישירות ללא הצגת השינויים מראש	אי-הכללת כלי <code>write/edit</code> ברשימת <code>tools</code> . אם נדרשת כתיבה – דרישה לאישור משתמש מפורש
3. חשיפת מידע רגיש	פקודה כמו <code>ls -la /etc/</code> עשויה לחשוף מידע תצורה רגיש שלא צריך להיות נגיש	שימוש ב- <code>bashoutput</code> במקום <code>bash</code> כאשר די בקריאה. הגבלת <code>bash</code> למקרים חריגים בלבד
4. הרצת פקודות מנהלתיות	הסוכן מנסה להריץ פקודה מנהלתית כמו <code>systemctl restart apt-get</code> או <code>install</code>	הרצת הסוכן עם משתמש מוגבל (לא <code>root</code>), חסימת <code>sudo</code> בהנחיות <code>sys-tem_prompt</code>

החוכמה האמיתית טמונה לא רק בהגבלת הכלים הזמינים, אלא גם בהטמעת תרבות אבטחה בתוך ה-`system_prompt` עצמו. הסוכן צריך "להבין" שהוא פועל בסביבה מבוקרת, וכל פעולה מערכתית צריכה להיות מוצדקת ומאושרת.

7.8 ההגדרה המעשית: הגבלת הרשאות ב-YAML

כעת, לאחר שהבנו את הסיכונים והעקרונות, ניישם אותם במעשה. נרחיב את קובץ התצורה של סוכן ה-expert-analyst שלנו (expert-analyst.yaml) כדי לכלול את כלי bash ו-bashoutput, תוך הקפדה על הטמעת הנחיות אבטחה מחייבות.

7.8.1 עדכון קובץ ה-YAML: הוספת כלי Bash מבוקרים

נפתח את קובץ התצורה agents/expert-analyst.yaml ונעדכן את רשימת הכלים וה-system_prompt:

YAML - קובץ expert-analyst.yaml מעודכן עם אבטחת Bash

```
# () ווכדע Knowledge-Agent-Yoram/agents/expert-analyst.yaml
---
name: "expert-analyst"
description: "מסיכמסמחותינג, תמדקתמסינותנתפילשלדעויהמהחמומלכוס"
מוכיסוסיינכט
".טלחומטיעוצקמקוידלעלערומשלגביחממהזלכוסל.פיקמלעדיתמ"
model: "opus"
tools:
  - read
  - websearch
  - grep
  - structured-report
  - bashoutput # חוטב טלפל אלמ bash ינפ לע פדעומ
  - bash # תוריהזב דימת דא, תלבגומ תושימג דרוצל
  # 'edit' וא 'write' תפסוהמ סיענמן

# --- (RAG) ירוטקו לורכז תורדגה ---
vector_memory:
  provider: "qdrant" # ירוטקווה סינותנה דסמ קפס
  url: "http://localhost:6333" # דסמה לש Endpoint-ה תבותכ
  embedding_model: "nomic-embed-text" # מירוטקול טסקט תרמהל שמשמה לדומה

knowledge:
  retriever: "mcp" # עדי תפילשל לוקוטורפה (Model Context Protocol)
  mcp_url: "http://localhost:5100" # ימוקמה Retriever-ה תרש תבותכ
  collection: "project_docs" # הפילשל (סיכמסמה) רגאמ היצקלוקה שס

system_prompt: |
  עדימה לע ססבתהב קמונמו תמואמ תויהל בייח טלפ לכ. קיודמו ריכב טסילנא התא
  פלשנ וא קפוסש.
  תימשרו תיעוצקמ, תיתטיש תויהל תבייח דלש השיגה. תויצלוקפסמ ענמיה

  **רתומ `bash`-ו `bashoutput` ילכב שומישה **:תבייחמ החטבא **תארוה
  **קרו
  לע עדימ תלבק וא (סיאתמ וניא `read`) רשאכ קבוק לכות תגצה, לוחבא תורטמל
  הביבסה.
  תא וא סיצבקה תכרעמ תא תונשמה תודוקפ תצרהל ולא סילכב שמתשהל ****רוסא
  .(סיינוציח `rm`, `mv`, `write`, `edit`) לוגכ תכרעמה תייצרוגיפנוק

  **סיפסונ תוחיטב **יללכ
  - להנמ תואשרה וא `sudo` סע תודוקפ קירת לא סלועל.
  - טלפ תאירקב יד רשאכ `bash` ינפ לע `bashoutput`-ב שומיש פדעה.
  - תיתכרעמ הדוקפ עוציב ינפל שמתשמה הרהבה שקב - קפס לש הרקמ לכב.
```

7.8.2 ניתוח ההגדרות: שכבות האבטחה

נבחן את מנגנוני האבטחה שהטמענו בקובץ זה:

שכבה ראשונה: בחירת כלים

- הוספנו במפורש את `bash` ו-`bashoutput`, אך **השארנו מחוץ לרשימה** את `write`, `edit`, `notebookedit` (ככל שהם קיימים כ-`built-in`). זהו יישום ישיר של עיקרון ההרשאה המינימלית: אם הכלי אינו נדרש להשגת המטרה, הוא לא צריך להיות זמין.

- העדפת `bashoutput` מעידה על מדיניות "קריאה בלבד" כברירת מחדל.

שכבה שנייה: הנחיית אבטחה ב-`system_prompt`

ה-`system_prompt` משמש כמעצור פנימי עבור המודל. הוספת הנחיה מפורשת ומפורטת מספקת הדרכה ברורה לסוכן כיצד להשתמש באופן אחראי בכלים החזקים הללו. חשוב להבין: הנחייה זו **אינה תחליף לאמצעי אבטחה טכניים** (כמו הרצת הסוכן במשתמש מוגבל), אך היא מהווה שכבת הגנה משמעותית מפני שימוש לא מכוון או מוטעה.

שכבה שלישית: הנחיות התנהגותיות ספציפיות

הכללים הנוספים שהוספנו מספקים מדריך מעשי:

- אי-שימוש ב-`sudo` – מונע ניסיונות להרחיב הרשאות.

- העדפת `bashoutput` – מדגיש את מדיניות הזהירות.

- בקשת הבהרה במקרה של ספק – מעודד שקיפות ומניעת טעויות.

7.8.3 דוגמאות שימוש: הגוון בין בטוח למסוכן

כדי להמחיש את ההבדל בין שימוש ראוי למסוכן, נבחן מספר תרחישים:

דוגמה 1: שימוש בטוח – אבחון מערכת

בקשת משתמש: "הצג לי את שימוש הדיסק הנוכחי והזיכרון הפנוי במערכת."

תגובת הסוכן הצפויה:

תגובת הסוכן - שימוש בטוח ב-`tuftuohsab`

```
ינא
:ןוחבאל bashoutput ב שחתשנ

$ df -h
Filesystem      Size  Used Avail Use% Mounted on
/dev/sda1        50G   32G   16G   67% /

$ free -h
              total        used        free      shared  buff/cache
Mem:          15Gi         8.2Gi         3.1Gi         256Mi         4.0Gi
```

כאן, הסוכן ביצע פקודות קריאה בלבד, ללא שינוי במערכת.

דוגמה 2: שימוש בעייתי - נסיון כתיבה

בקשת משתמש: "עדכן את קובץ התצורה config.txt והוסף את השורה debug=true".
תגובת הסוכן הצפויה (מבוססת על `system_prompt`): אני מזהה שהבקשה דורשת שינוי בקובץ תצורה. על פי הנחיות האבטחה שלי, אני לא מורשה לבצע כתיבה או עריכה ישירה של קבצים.

האם תרצה שאציג לך את תוכן הקובץ הנוכחי, כך שתוכל לבצע את השינוי בעצמך?
לחלופין, אוכל להדריך אותך כיצד לעשות זאת בצורה בטוחה.

כאן, הסוכן **סירב** לבצע את הפעולה, ובמקום זאת הציע חלופות בטוחות.

דוגמה 3: שימוש זהיר - קריאת קובץ מיוחד

בקשת משתמש: "הצג לי את תוכן קובץ ה-log האחרון ב-/var/log/".

תגובת הסוכן הצפויה:

תגובת הסוכן - שימוש מוגבל ב-hsab

```
ינא
ותגצהל read-b וכמ רחאלו קבוקה רותיאל bash-b שמתשמ

$ ls -lt /var/log/*.log | head -1
-rw-r--r-- 1 root root 45123 Nov 17 14:22 /var/log/syslogטעכ

read. תועצמאב קבוקה לכות תא ארוק ינא
```

כאן, הסוכן השתמש ב-bash לצורך מציאת הקובץ (פעולה קריאה), ואז עבר לכלי read המאובטח יותר להצגת התוכן.

7.8.4 מבחן העקרונות: אימות התנהגות

לאחר עדכון הקובץ, מומלץ לבצע מבחן פשוט כדי לוודא שהסוכן מכבד את הנחיות האבטחה:

ברסל דירצ - הקיחמ ווי

```
טסיינמזהטסיצבקהטלכטאטקחמ"
e agent run expert-analyst --task
```

מבחן 1: נסיון מחיקה

תגובה צפויה: הסוכן צריך לסרב או לבקש הבהרה, ולא לבצע מחיקה אוטומטית.

תרומ - עד

```
le agent run expert-analyst --task "תכרעמבתנקתומהPythonטטרגהמ"
```

מבחן 2: בקשת מידע

תגובה צפויה: הסוכן יריץ `python --version` או דומה ויציג את התוצאה - פעולה בטוחה.

7.8.5 מגבלות והסתייגויות

חשוב להדגיש כמה נקודות מפתח:

1. ה-`system_prompt` אינו מנגנון אבטחה מוחלט. מודלי AI עלולים לפרש הנחיות בצורה שגויה, במיוחד כאשר הבקשה מהמשתמש מעורפלת. לכן, אמצעי אבטחה טכניים (הרצה במשתמש מוגבל, סביבות `sandbox`, ניטור פעילות) הם הכרחיים בסביבת ייצור.
2. אין תחליף לביקורת אנושית. במערכות קריטיות, כל פעולה שמשנה מצב (גם אם היא מותרת) צריכה לעבור ביקורת או לדרוש אישור מפורש.
3. עדכון מתמיד של המדיניות. ככל שהסוכן מתפתח ומקבל יכולות נוספות, יש לעדכן גם את הנחיות האבטחה ב-`system_prompt` בהתאם.

7.8.6 סיכום ומסקנות

בפרק זה למדנו כי כוח גדול מחייב אחריות גדולה. כלי ה-`bash`, למרות היותו חיוני למשימות רבות, דורש ניהול זהיר ומושכל. עיקרון ההרשאה המינימלית אינו רק עיקרון תיאורטי – הוא צריך להתבטא בכל שכבה: בבחירת הכלים הזמינים, בניסוח הנחיות ברורות ב-`system_prompt`, ובהטמעת תרבות אבטחה כוללת בסביבת העבודה. סוכן ה-`expert-analyst` שלנו, "יורם", כעת מצויד בכלי `bash` מבוקר, המאפשר לו לבצע אבחונים מערכתיים הכרחיים, תוך שמירה על שלמות וביטחון המערכת. בפרק הבא (פרק 9), נריץ את הסוכן המלא ונבחן כיצד כל היכולות שבנינו – `RAG`, `SKILL` מותאם, אינטגרציה עם `Python`, וכעת גם `bash` מאובטח – פועלות יחד בהרמוניה.

7.8.7 נספח: רשימת פקודות מסוכנות לחסימה

להלן רשימה חלקית של פקודות שכדאי לחסום במפורש (אם לא באמצעים טכניים, לפחות בהנחיות system_prompt). טבלה 4 מפרטת את הפקודות המסוכנות ביותר ואת הסיבות לחסימתן:

טבלה 4: פקודות מסוכנות ודרכי המניעה

מניעה	מדוע מסוכנת	פקודה
חסימה מוחלטת בהנחיות	מוחקת קבצים, עלולה למחוק נתונים קריטיים	rm -rf
חסימה מוחלטת בהנחיות	מעבירה או מחליפה קבצים, עלולה לשבש מבנה תיקיות	mv
חסימה מוחלטת בהנחיות	משנה הרשאות, עלולה לחשוף קבצים רגישים או לשבש גישה	chmod, chown
דרישת משתמש לא-root, חסימה מפורשת ב-system_prompt	מריצה בהרשאות מנהל, נותנת גישה בלתי מוגבלת	sudo
חסימה מוחלטת (העדפה לכלי write מבוקר אם כלל)	כותבת תוכן לקובץ, עלולה לדרוס תצורה או קוד	echo > file
חסימה מוחלטת בהנחיות	מורידה ומריצה קוד חיצוני, פוטנציאל ל-malware	curl bash
חסימה מוחלטת בהנחיות	מתקינה תוכנות, משנה את המערכת	apt-get, yum
חסימה מוחלטת בהנחיות	משנה תצורת רשת או קישוריות	iptables, ifconfig

המלצה מעשית: במקום רשימת חסימה (blacklist), כדאי לאמץ גישת רשימה מותרת (whitelist) – רק פקודות ספציפיות ומאושרות יורשו לרוץ, וכל השאר ייחסמו כברירת מחדל.

9 הרצה ובדיקה

הגענו לשלב המכריע – הפעלת הסוכן המותאם שבנינו ובדיקת יכולותיו בפועל. לאחר שמונה פרקים שבהם הגדרנו, תכננו והרכבנו את expert-analyst, הגיע הזמן לראות אותו פועל בסביבה אמיתית. זהו הרגע שבו המושגים העיוניים הופכים למציאות מוחשית, והעקרונות הפילוסופיים שדנו בהם מקבלים ביטוי מעשי.

בפרק זה נלמד כיצד להפעיל את הסוכן באמצעות Claude CLI, כיצד לבצע בדיקות שיטתיות על כל רכיב ורכיב, וכיצד לוודא שהסוכן שלנו עומד בציפיות ופועל כמתוכנן. נדגים תרחישי שימוש מלאים, נפתור בעיות נפוצות, ונצייד אתכם בכלים הנדרשים להמשך הדרך.

9.1 הפעלת הסוכן: הפקודה claude agent run

הדרך הבסיסית ביותר להפעיל סוכן מותאם היא באמצעות הפקודה claude agent run. זוהי הנקודה שבה כל העבודה שביצענו מתחברת לכלל אחד מלוכד – הקובץ expert-analyst.yaml שיצרנו, מערך ה-SKILLS שפיתחנו, והגדרות ה-RAG שהתקנו, כולם מתאחדים לישות אחת פעילה.

9.1.1 תחביר הפקודה הבסיסית

הפקודה הפשוטה ביותר להפעלת סוכן היא:

Basic agent invocation - Bash command

```
# Navigate to project directory
cd ~/Knowledge-Agent-Yoram

# Run the expert-analyst agent
claude agent run expert-analyst

# Or specify path explicitly
claude agent run agents/expert-analyst.yaml
```

כאשר מריצים את הפקודה הזו, Claude CLI מבצע את השלבים הבאים:

- 1. טעינת קובץ ה-YAML:** המערכת קוראת את הקובץ agents/expert-analyst.yaml, מנתחת את המבנה שלו ומזהה את כל השדות המוגדרים – השם, המודל, הכלים, ה-system_prompt, וההגדרות המתקדמות כמו vector_memory ו-knowledge.
- 2. אתחול זיכרון ווקטורי:** אם הוגדר vector_memory, המערכת מנסה להתחבר למסד הנתונים הווקטורי (במקרה שלנו – Qdrant על http://localhost:6333). אם החיבור נכשל, תופיע אזהרה, אך הסוכן ימשיך לפעול ללא יכולות RAG.

3. **טעינת SKILLS:** המערכת סורקת את התיקיה /skills ומזהה את ה-SKILL בשם structured-report. היא קוראת את קובץ ה-SKILL.md, מנתחת את ה-input_schema, ומכינה את הסוכן לקבל קריאות ל-SKILL זה.
4. **אתחול מודל opus:** המערכת פונה ל-API של Anthropic ומבצעת אתחול של מודל Claude Opus, המודל החזק ביותר הזמין. זה המקום שבו השימוש במפתח ה-API שהגדרנו בפרק 1 מתממש.
5. **הצגת ממשק אינטראקטיבי:** לאחר שכל הרכיבים אותחלו בהצלחה, מוצג לנו ממשק שורת פקודה אינטראקטיבי שבו נוכל לשוחח עם הסוכן, להעביר לו משימות, ולקבל ממנו תגובות מפורטות.

9.1.2 פרמטרים נוספים ומתקדמים

ניתן להעביר פרמטרים נוספים לפקודה כדי לשלוט בהתנהגות הסוכן:

Advanced agent run parameters - Bash

```
# Run with specific task (non-interactive)
claude agent run expert-analyst --task "Analyze the project documentation"

# Run with custom model override
claude agent run expert-analyst --model sonnet

# Run with verbose logging
claude agent run expert-analyst --verbose

# Run with specific API key profile
ANTHROPIC_API_KEY=sk-ant-dev-key claude agent run expert-analyst

# Combine parameters
claude agent run expert-analyst \
  --task "Summarize the latest changes" \
  --model opus \
  --verbose
```

הפרמטר task – מאפשר לנו להפעיל את הסוכן עם משימה ספציפית ולצאת לאחר ביצועה, במקום להיכנס למצב אינטראקטיבי. זה שימושי במיוחד כאשר רוצים לשלב את הסוכן בתהליכי אוטומציה או בסקריפטים.

9.2 בדיקת כלים מובנים

השלב הראשון בתהליך הבדיקה הוא לוודא שהכלים המובנים (grep, websearch, read) פועלים כראוי. כל כלי צריך להיבדק בנפרד כדי לוודא שאין בעיות חיבור, הרשאות, או

9.2.1 בדיקת כלי read

נבדוק שהסוכן מסוגל לקרוא קבצים מתיקיית הפרויקט:

Testing read tool - Interactive session

```
User: Please read the file agents/expert-analyst.yaml and tell me what
model is configured.

Expert-Analyst: I'll read the agent configuration file.

[Agent uses read tool]

Based on the file `agents/expert-analyst.yaml`, the configured model is
**opus**.
This is the most powerful model in the Claude family, suitable for
complex analytical
tasks requiring deep reasoning and comprehensive understanding.

The full configuration shows:
- Model: opus
- Tools: read, websearch, grep, structured-report, bashoutput, bash
- Vector memory enabled with Qdrant
- Knowledge retriever via MCP protocol
```

אם הכלי read פועל כראוי, הסוכן יוכל לקרוא את הקובץ, לנתח את תוכנו, ולהשיב לנו בצורה מדויקת. אם יש בעיה (למשל, הקובץ לא קיים או אין הרשאות קריאה), הסוכן יודיע לנו על כך במפורש.

9.2.2 בדיקת כלי websearch

כעת נבדוק שהסוכן מסוגל לחפש מידע באינטרנט: בדיקה זו מאמתת שמפתח ה-API פעיל, שהחיבור לאינטרנט תקין, ושהסוכן מסוגל לבצע חיפושים ולעבד את התוצאות בצורה משמעותית.

9.2.3 בדיקת כלי grep

נבדוק חיפוש טקסט בתוך קבצי הפרויקט: כלי grep חיוני למציאת מידע בפרויקטים גדולים. בדיקה זו מאמתת שהסוכן יכול לסרוק קבצים, למצוא דפוסי טקסט, ולהציג תוצאות רלוונטיות.

Testing websearch tool - Interactive session

User: Search **for** the latest information about Anthropic Claude models **and** summarize the key improvements **in** recent versions.

Expert-Analyst: I'll search for recent information about Claude models.

[Agent uses websearch tool]

Based on my web search, here are the key improvements in recent Claude versions:

- Claude 3 Family (March 2024):**
 - Three models: Opus (most powerful), Sonnet (balanced), Haiku (fastest)
 - Improved reasoning capabilities
 - Better multilingual support
- Extended Context Window:**
 - Increased to 200K tokens in Claude 3
 - Enables processing of entire books or large codebases
- Vision Capabilities:**
 - Can now analyze images and diagrams
 - Useful for document analysis and visual data processing

These improvements make Claude particularly suitable for knowledge management and analytical tasks like those in our expert-analyst agent.

Testing grep tool - Interactive session

User: Search **for all** occurrences of "vector_memory" in the project files .

Expert-Analyst: I'll search for "vector_memory" across the project.

[Agent uses grep tool]

Found 3 occurrences of "vector_memory":

1. **agents/expert-analyst.yaml** (line 16):
Configuration section defining the Qdrant vector database connection.

2. **skills/structured-report/SKILL.md** (line 45):
Reference to using vector memory for historical data retrieval.

3. **docs/rag-setup.md** (line 12):
Documentation explaining the vector_memory configuration parameters.

All occurrences are properly configured and consistent across the project.

9.3 בדיקת SKILLs מותאמים

לאחר שוידאנו שהכלים המובנים פועלים, עלינו לבדוק את ה-SKILL המותאם שיצרנו – structured-report. בדיקה זו מורכבת יותר מכיוון שהיא דורשת נתוני קלט, עיבוד מורכב, ופלט מובנה.

9.3.1 הכנת נתוני בדיקה

ראשית, נכין קובץ CSV פשוט עם נתונים לדוגמה:

Example test data - CSV file (scripts/test_data.csv)

```
date,product,sales,revenue
2024-01-15,Widget A,120,3600.00
2024-01-16,Widget B,85,2550.00
2024-01-17,Widget A,150,4500.00
2024-01-18,Widget C,200,8000.00
2024-01-19,Widget B,95,2850.00
2024-01-20,Widget A,180,5400.00
```


9.3.2 הפעלת ה-SKILL עם נתונים אמיתיים

כעת נבקש מהסוכן לנתח את הנתונים באמצעות ה-SKILL שיצרנו: תגובה זו מדגימה את הערך האמיתי של SKILL מותאם. הסוכן לא רק קרא את הקובץ – הוא עיבד את הנתונים, ביצע חישובים סטטיסטיים, ארגן את המידע בטבלה מסודרת, והציג תובנות משמעותיות. כל זה נעשה בעקבות ההוראות שהגדרנו בקובץ SKILL.md.

9.3.3 בדיקת טיפול בשגיאות

חשוב גם לבדוק איך ה-SKILL מתנהג כאשר מתקבלים נתונים שגויים או חסרים: טיפול נכון בשגיאות הוא סימן לסוכן בשל ומקצועי. הסוכן שלנו לא קרס, לא החזיר הודעת שגיאה טכנית סתמית, אלא הסביר בבירור מה הבעיה, הציע פתרונות אלטרנטיביים, והמשיך לפעול בצורה יציבה.

9.4 אימות שליפת RAG

אחד הרכיבים המתקדמים ביותר בסוכן שלנו הוא מערך ה-RAG (Retrieval-Augmented Generation) שמאפשר לו לשלוף ידע ממאגר מסמכים ווקטורי. בדיקת רכיב זה מורכבת יותר מכיוון שהיא דורשת תשתית חיצונית פעילה (Qdrant ושרת MCP).

9.4.1 הכנת סביבת הבדיקה

לפני שנבדוק את ה-RAG, עלינו לוודא שהתשתית הדרושה פעילה:

9.4.2 בדיקת שליפת מידע מהזיכרון הווקטורי

כעת נבקש מהסוכן לשלוף מידע שאינו זמין בהקשר המיידי שלו, אלא רק במאגר הידע: הבדיקה הזו מאמתת שלושה דברים קריטיים: (א) המערכת הווקטורית פעילה ומקבלת שאילתות. (ב) המסמכים הוטמעו (embedded) כראוי ונשמרים בקולקציה הנכונה. (ג) הסוכן מסוגל לשלוף מידע רלוונטי, לסנתז אותו, ולהציג אותו בצורה מובנית עם הפניות למקורות.

9.4.3 בדיקת שאילתות מורכבות

נבדוק שאילתה מורכבת יותר הדורשת שילוב של מספר מקורות: בדיקה זו מדגימה את עוצמתו האמיתית של RAG. הסוכן לא רק שלף מידע מקטעים בודדים – הוא איחד מידע ממספר מקורות שונים (פרקים 4 ו-5, קובץ החלטות עיצוב), ניתח את ההבדלים, והציג סינתזה מקיפה עם המלצה מושכלת. זה בדיוק מה שרצינו להשיג כאשר תכננו את expert-analyst – סוכן שמסוגל לעבוד עם בסיס ידע רחב מעבר לחלון ההקשר המיידי שלו.

Testing structured-report SKILL - Interactive session

User: Use the structured-report SKILL to analyze scripts/test_data.csv.
I need a summary analysis showing total sales **and** revenue by product.

Expert-Analyst: I'll analyze the sales data using the structured-report SKILL.

[Agent invokes structured-report SKILL with parameters:

```
file_path: "scripts/test_data.csv"
analysis_type: "summary"
parameters: {"group_by": "product"}
]
```

Structured Sales Report

Summary Analysis

Total Sales by Product:

Product	Total Units	Total Revenue	Avg Daily Sales
Widget A	450	\$13,500.00	150.0
Widget B	180	\$5,400.00	90.0
Widget C	200	\$8,000.00	200.0
Total	**830**	**\$26,900.00**	**138.3**

Key Findings:

- Widget A leads in total sales (450 units, 54.2% of total)
- Widget C has highest daily average despite fewer days
- Total revenue: \$26,900.00 over 6-day period
- Average daily revenue: \$4,483.33

Data Quality:

- 6 records processed
- 0 errors detected
- Date range: 2024-01-15 to 2024-01-20

Testing error handling - Interactive session

User: Try to use structured-report on a **file** that doesn't exist:
scripts/missing_file.csv

Expert-Analyst: I'll attempt to process the specified **file**.

[Agent invokes structured-report SKILL]

****Error Report:****

I encountered an issue **while** trying to analyze `scripts/missing_file.csv`
`:

- ****Error Type:**** File Not Found
- ****Details:**** The specified **file** does **not** exist **in** the project directory
- ****Suggestion:**** Please verify the **file** path. Available files **in** scripts/:
 - scripts/test_data.csv
 - scripts/data_processor.py
 - scripts/test_agent.py

Would you like me to analyze one of the existing files instead?

Verify RAG infrastructure - Bash commands

```
# Check if Qdrant is running
curl http://localhost:6333/health

# Expected response:
# {"status":"ok"}

# Check if MCP server is running
curl http://localhost:5100/health

# Expected response:
# {"status":"ready","collections":["project_docs"]}

# If not running, start the services:
docker start qdrant
python scripts/mcp_server.py &
```

Testing RAG retrieval - Interactive session

User: Based on the project documentation, what are the security best practices mentioned **for** using the bash tool?

Expert-Analyst: I'll search the knowledge base for security best practices.

```
[Agent queries vector memory for "bash security best practices"]  
[Retrieved 3 relevant chunks from project_docs collection]
```

Based on the project documentation, here are the key security best practices for bash tool usage:

****1. Principle of Least Privilege:****

- Grant bash access only when absolutely necessary
- Prefer bash output for read-only operations

****2. Prohibited Commands:****

- Never use: rm, mv, sudo, system configuration changes
- Avoid: write operations to critical directories

****3. System Prompt Guardrails:****

- Always include explicit security instructions
- Document prohibited operations clearly

****4. Validation Before Execution:****

- Request user confirmation for system commands
- Log all bash executions for audit trail

****Source References:****

- docs/security-guidelines.md (Section 3.2)
- agents/expert-analyst.yaml (system_prompt_lines 32-41)
- examples/secure-bash-examples.md

These practices ensure the agent operates safely while maintaining necessary flexibility for diagnostic tasks.

Complex RAG query - Interactive session

User: Compare the approach to custom SKILLs **in** this project versus using built-**in** tools only. What are the advantages mentioned **in** the documentation?

Expert-Analyst: I'll retrieve and compare information from multiple sources.

[Agent performs semantic search across documentation]

[Combines information from chapters 4 and 5]

****Comparison: Built-in Tools vs Custom SKILLs****

****Built-in Tools Approach:****

- Limited to predefined functionality (read, websearch, grep)
- No structured input validation
- Generic output format
- Suitable for simple, one-off tasks

****Custom SKILL Advantages (per documentation):****

1. ****Structured Input Schema:****

- ▮ JSON Schema validation ensures correct parameters
- ▮ Self-documenting interface
- ▮ Reduces user errors

2. ****Explicit Workflow Instructions:****

- ▮ Step-by-step execution guidance
- ▮ Consistent output format
- ▮ Reproducible results

3. ****Modularity and Reuse:****

- ▮ SKILL can be shared across projects
- ▮ Version-controlled improvements
- ▮ Team collaboration enabled

4. ****Computational Augmentation:****

- ▮ Can invoke Python scripts (e.g., data_processor.py)
- ▮ Access to scientific libraries
- ▮ Complex data transformations possible

****Recommendation from docs:****

Use built-in tools for exploration and discovery. Create custom SKILLs for recurring, complex analytical tasks that require structured input and consistent output.

****Sources****

- Chapter 4: Built-in Tools (pages 45-52)

- Chapter 5: Custom SKILL Development (pages 63-71)

- docs/design-decisions.md (Section 2.3)

9.5 הדגמת תהליך עבודה מלא End-to-End

כעת, לאחר שבדקנו כל רכיב בנפרד, הגיע הזמן להדגים תהליך עבודה מלא שמשלב את כל היכולות של הסוכן – כלים מובנים, SKILL מותאם, שלפת RAG, ואפילו שימוש מבוקר ב-bash. זה הרגע שבו כל מה שבנינו לאורך הספר מתאחד לכדי חוויה אחת מלוכדת.

9.5.1 תרחיש הדגמה: ניתוח מקיף של פרויקט

נדמיין משימה אמיתית: מנהל פרויקט רוצה דוח מקיף על מצב הפרויקט, כולל ניתוח קבצי קוד, נתוני מכירות, ומידע מהתיעוד ההיסטורי. נבקש מהסוכן לבצע את המשימה הזו בשלמותה:

דוח זה הוא הגלמה מושלמת של מה שסוכן AI מותאם יכול להשיג. הוא לא רק ענה על השאלה – הוא ארגן את התשובה בצורה מובנית, שילב מידע ממקורות מרובים, ביצע ניתוחים כמותיים ואיכותיים, הציג המלצות מעשיות, ואף סיפק תיעוד מפורט של המקורות שבהם השתמש. זהו בדיוק הסוג של עבודה שמנהל פרויקט או אנליסט בכיר היה עושה, אך הסוכן עשה זאת בשניות ספורות עם דיוק מוחלט.

9.5.2 ניתוח התהליך

בואו נפרק את התהליך שהסוכן ביצע:

שלב 1 – קריאת נתוני מכירות: הסוכן זיהה שיש צורך לנתח קובץ CSV והפעיל באופן אוטומטי את ה-SKILL בשם structured-report. הוא העביר את הפרמטרים הנכונים (נתיב הקובץ, סוג הניתוח) והמתין לתוצאות.

שלב 2 – סריקת שינויים בקוד: הסוכן הבין שכדי למצוא שינויים אחרונים, עליו להשתמש ב-bash. הוא הריץ פקודת find בטוחה (קריאה בלבד, ללא שינוי קבצים) ואסף רשימת קבצים שהשתנו בשבוע האחרון.

שלב 3 – קריאת תוכן הקבצים: עבור כל קובץ שזוהה, הסוכן השתמש בכלי read כדי לקרוא את התוכן ולבצע סקירת קוד בסיסית (בדיקת ציות ל-PEP 8, ספירת שורות, זיהוי שינויים עיקריים).

שלב 4 – שאילתת בסיס הידע: הסוכן ביצע חיפוש סמנטי במערכת ה-RAG אחר "best practices security bash", שלף את המסמכים הרלוונטיים, והשווה את הממצאים מהקוד למדיניות הארגונית.

שלב 5 – סינתזה ודיווח: הסוכן שילב את כל המידע לדוח אחד מובנה, ארגן אותו לפי קטגוריות לוגיות (ביצועים, קוד, תאימות, סיכונים, המלצות), והציג אותו בפורמט מקצועי עם תיעוד מקורות.

כל שלב הזה התבצע באופן אוטונומי, ללא צורך בהתערבות האדם. הסוכן קיבל מטרה רחבה ("הכן דוח מקיף") ותרגם אותה לסדרה של פעולות קונקרטיות, תוך שימוש נבון בכל הכלים העומדים לרשותו.

Full workflow demonstration - Interactive session

User: I need a comprehensive project status report that includes:

1. Analysis of the latest sales data (scripts/test_data.csv)
2. Summary of recent code changes **in** the project
3. Comparison **with** best practices **from** our knowledge base
4. Risk assessment **and** recommendations

Expert-Analyst: I'll prepare a comprehensive project status report combining multiple data sources and analysis methods.

[Step1: Analyze sales data using structured-report SKILL]
Analyzing sales data from scripts/test_data.csv...

[Step2: Search for recent file changes using bash]
\$ find . -name "*.py" -mtime -7 -type f
./scripts/data_processor.py
./scripts/test_agent.py

[Step3: Read code changes using read tool]
Reading modified files...

[Step4: Query knowledge base for best practices]
Searching vector memory for relevant best practices...

[Synthesis and Report Generation]

=====PROJECT STATUS REPORT=====

Generated: 2024-11-17 17:30:00

Agent: expert-analyst v1.0

1. SALES PERFORMANCE ANALYSIS

Summary (Past 6 Days):
- Total Revenue: \$26,900.00
- Total Units Sold: 830
- Average Daily Revenue: \$4,483.33
- Top Product: Widget A (54.2% of sales)

Trend Assessment:
- Positive: Consistent sales across all products
- Concern: Widget C limited to single day (need more data)
- Recommendation: Expand Widget C availability

2. RECENT CODE CHANGES

Modified Files (Past 7 Days):
1. **data_processor.py** (Updated 2024-11-15)
- Added error handling for missing columns
- Improved statistical analysis functions
- Code quality: PEP8 compliant, <150 lines

9.6 פתרון בעיות נפוצות

למרות התכנון המוקפד והבדיקות השיטתיות, ייתכנו מצבים שבהם הסוכן לא יפעל כצפוי. בחלק זה נסקור בעיות נפוצות ואת הפתרונות להן.

9.6.1 שגיאה: חיבור למסד הנתונים הווקטורי נכשל

Error: Vector database connection failed

```
Error: Failed to connect to Qdrant at http://localhost:6333
Cause: Connection refused
```

Troubleshooting steps:

1. Verify Qdrant **is** running:

```
$ docker ps | grep qdrant
```

2. Check Qdrant logs:

```
$ docker logs qdrant
```

3. Restart Qdrant:

```
$ docker restart qdrant
```

4. Verify URL **in** agent YAML:

```
vector_memory:
```

```
  url: "http://localhost:6333" # Must match running instance
```

5. Test connection manually:

```
$ curl http://localhost:6333/health
```


9.6.2 שגיאה: SKILL לא נמצא

Error: SKILL not found

Error: SKILL 'structured-report' **not** found **in** project

Troubleshooting steps:

1. Verify SKILL directory exists:

```
$ ls -la skills/structured-report/
```

2. Check SKILL.md presence:

```
$ cat skills/structured-report/SKILL.md
```

3. Validate YAML frontmatter:

```
name: "structured-report" # Must match the SKILL name used in agent
```

4. Check agent YAML tools **list**:

```
tools:
```

```
- structured-report # Must be listed here
```

5. Ensure correct working directory:

```
$ pwd # Should be project root, not subdirectory
```

9.6.3 שגיאה: מודל לא זמין

Error: Model not available

Error: Model 'opus' **not** available **for** your account

Possible causes **and** solutions:

1. ****API Key Issue:****

```
$ echo $ANTHROPIC_API_KEY # Verify key is set
```

2. ****Model Access:****

```
- opus requires paid account
```

```
- Try: model: "sonnet" (free tier available)
```

3. ****Rate Limiting:****

```
- Wait a few minutes and retry
```

```
- Check usage at https://console.anthropic.com/
```

4. ****Temporary Override:****

```
$ claude agent run expert-analyst --model sonnet
```

Issue: Agent responds slowly

Symptom: Agent takes 30+ seconds to respond

Diagnosis **and** optimization:

1. ****Check model choice:****
 - opus: Most powerful, slowest (~10-30s)
 - sonnet: Balanced (~5-10s)
 - haiku: Fastest (~1-3s)

Switch to faster model **if** speed critical.
2. ****Vector memory overhead:****
 - RAG queries add latency
 - Disable **if not** needed **for** current task:


```
$ claude agent run expert-analyst --no-rag
```
3. ****Network latency:****

```
$ ping api.anthropic.com # Check connectivity
```
4. ****Large context:****
 - Reduce **input** size
 - Split large tasks into smaller chunks
5. ****Enable verbose mode to identify bottleneck:****

```
$ claude agent run expert-analyst --verbose
```

9.7 סיכום והמשך הדרך

הגענו לסיום המסע. על פני תשעה פרקים, עברנו יחד דרך ארוכה – מהבנת העקרונות הפילוסופיים של סוכנים אוטונומיים, דרך התקנת Claude CLI ויצירת מבנה פרויקט, הגדרת זהות הסוכן ויכולותיו, בניית SKILLS מותאמים אישית, שילוב עיבוד Python מתקדם, הטמעת מערכת RAG לניהול ידע, אבטחת הכלים המערכתיים, ועד להרצה ובדיקה מקיפה של הסוכן המושלם.

הסוכן expert-analyst שבנינו אינו רק דוגמה טכנית – הוא ישות דיגיטלית מלאה יכולת שמסוגלה לבצע משימות אנליטיות מורכבות, לשלוף מידע ממקורות מגוונים, לעבד נתונים בצורה מתמטית מדויקת, ולהציג תובנות במבנה מקצועי. והכל – תוך שמירה על עקרונות אבטחה קפדניים, תיעוד מלא של מקורות המידע, וטיפול נכון בשגיאות.

9.7.1 מה למדנו?

למהנדסי התוכנה הבכירים ביניהם: הבנתם כיצד לתכנן ארכיטקטורה מודולרית של סוכן AI, איך לבנות ממשק מוגדר (input_schema) ליכולות מותאמות, וכיצד לשלב סוכנים בתהליכי עבודה אוטומטיים. העקרונות שלמדתם כאן – הפרדת דאגות, ממשקים מובנים, טיפול בשגיאות – הם אותם עקרונות שהופכים כל מערכת תוכנה לאמינה וניתנת לתחזוקה.

לחוקרים והאקדמאים ביניהם: חקרנו את התיאוריה מאחורי Retrieval-Augmented Generation, הבנו את המגבלות של חלונות הקשר במודלי שפה גדולים, ולמדנו כיצד שילוב של זיכרון ווקטורי עם מודל גנרטיבי יוצר מערכת חזקה מסכום חלקיה. שאלות המחקר שהעלינו – כיצד לבחור מודל embedding מתאים, איך לאזן בין שליפה רחבה לדיוק גבוה – הן אותן שאלות שעומדות בחזית המחקר ב-NLP כיום.

למתקדמים ולסטודנטים ביניהם: קיבלתם מפת דרכים ברורה וצעד-אחר-צעד לבניית סוכן AI משלכם. החל מההתקנה הראשונית, דרך יצירת קובץ YAML פשוט, ועד לבניית מערכת מלאה עם יכולות RAG – כל שלב היה מלווה בהסברים, דוגמאות קוד, ועצות לפתרון בעיות. אתם יוצאים מכאן עם הידע והביטחון לבנות סוכנים משלכם ולהתאים אותם לצרכים הספציפיים שלכם.

9.7.2 לאן מכאן?

ספר זה מסיים את הבסיס, אך הוא רק פותח את השער לעולם רחב של אפשרויות. הנה כמה כיוונים להמשך:

1. הרחבת SKILLS: הסוכן שלנו כרגע כולל רק SKILL אחד – structured-report. נסו לבנות SKILLS נוספים: code-analyzer לניתוח קוד, document-summarizer לסיכום מסמכים ארוכים, meeting-notes-generator לעיבוד פרוטוקולים. כל SKILL חדש מרחיב את היכולות של הסוכן ומאפשר לו להתמודד עם סוגים נוספים של משימות.

2. שיפור מערכת ה-RAG: הטמענו מסמכי תיעוד פרויקט בסיסיים בזיכרון הווקטורי. נסו להטמיע מקורות ידע נוספים – ספרים טכניים, מאמרים מדעיים, תקני תעשייה, פרוטוקולי ישיבות. ככל שבסיס הידע עשיר יותר, הסוכן מסוגל לספק תובנות עמוקות ומושכלות יותר. נסו גם מודלי embedding שונים (כמו OpenAI ada-002 או Cohere embed-v3) והשוו את איכות השליפה.

3. אינטגרציה עם מערכות ארגוניות: הסוכן שלנו עובד בסביבה מקומית מבוקרת. צעד הבא יכול להיות שילובו במערכות הארגון – חיבור למסד נתונים ארגוני (דרך SKILL מותאם עם SQL), אינטגרציה עם מערכות CRM או ERP, אוטומציה של דיווח שבועי למנהלים. זה הופך את הסוכן ממעבדה אקדמית לכלי עבודה יומיומי.

4. סוכנים מרובים בשיתוף פעולה: הרעיון המתקדם ביותר – במקום סוכן יחיד, בנו מערכת של סוכנים מיוחדים שעובדים יחד. סוכן אחד מתמחה בניתוח נתונים, אחר באיסוף מידע מהאינטרנט, שלישי בכתיבת קוד, ורביעי בניהול פרויקטים. הם מתקשרים זה עם זה, מעבירים תוצאות ביניהם, ומשלימים זה את זה. זהו העתיד של מערכות AI אוטונומיות.

9.7.3 מחשבות סיום

בפתח הספר דיברנו על המעבר מכלים פסיביים לסוכנים אקטיביים. ראינו כיצד טכנולוגיה שנראתה פעם כמדע בדיוני – מערכת שמבינה שפה טבעית, שולפת מידע ממקורות מרובים, מבצעת חישובים מורכבים, ומציגה תובנות משמעותיות – הפכה למציאות זמינה ומעשית שכל אחד מכם יכול לבנות.

אך חשוב לזכור: הסוכנים שבנינו הם כלים. הם מרחיבים את היכולות האנושיות שלנו, מאפשרים לנו לעבד כמויות מידע עצומות, לזהות דפוסים שהיו נעלמים מהעין, ולקבל החלטות מושכלות יותר. אבל השיקול, האתיקה, האחריות – אלה נשארים אנושיים לחלוטין. הסוכן מספק תובנות; אתם, המשתמשים והמפתחים, מחליטים מה לעשות איתן.

כשאתם ממשיכים מכאן ובונים סוכנים משלכם, זכרו את העקרונות שנלמדו: **תכננו בקפידה** (פרקים 0-2), **הגדירו זהות ברורה** (פרקים 3-4), **בנו יכולות מותאמות** (פרקים 5-6), **הרחיבו את הידע** (פרק 7), **אבטחו את המערכת** (פרק 8), **ובדקו בקפדנות** (פרק 9). סוכנים שנבנו על יסודות מוצקים אלה יהיו אמינים, שימושיים, ובטוחים.

בהצלחה בבניית הסוכנים שלכם. שיהיו חכמים, יעילים, ואחראיים. ושיעזרו לכם להשיג את המטרות שלכם – בין אם זה מחקר מדעי, ניהול פרויקטים, ניתוח עסקי, או כל תחום אחר שבו ידע, חשיבה ביקורתית, ויכולת ניתוח הם המפתח להצלחה.

"הטכנולוגיה הטובה ביותר היא זו שמעצימה בני אדם,
לא זו שמחליפה אותם."

— סוף פרק 9 —

— סוף הספר: בניית סוכן AI המותאם אישית —

9.8 English References

- 13 W. Stallings and L. Brown, *Computer Security: Principles and Practice*, 4th. Pearson, 2017.
- 14 J. H. Saltzer and M. D. Schroeder, "The protection of information in computer systems," *Proceedings of the IEEE*, vol. 63, no. 9, 1278–1308, 1975.
- 15 Anthropic. "Claude CLI official documentation. "[Online]. Available: [https : / / code.claude.com](https://code.claude.com)

10 רקע והשוואה ראשונית

בעולם פיתוח התוכנה המודרני, שלושה פורמטים נפוצים משמשים לצרכים שונים: YAML – פורמט סריאליזציה ידידותי למשתמש המיועד לקונפיגורציה ולהגדרות, JSON – פורמט העברת נתונים קל משקל בין מערכות, ו-Markdown – שפת סימון פשוטה לכתיבה ועיצוב טקסט. כל אחד מהפורמטים הללו נועד לפתור צרכים שונים, ולכל אחד מהם יתרונות וחסרונות ייחודיים.

נספח זה מציג השוואה מקיפה בין שלושת הפורמטים, תוך התמקדות בשימושים הרלוונטיים לפיתוח סוכני AI וכלי Claude CLI. ההשוואה כוללת טבלאות מפורטות, דוגמאות קוד מעשיות, והמלצות מבוססות שימוש.

10.1 השוואה ראשונית: YAML לעומת Markdown

טבלה 5 מציגה השוואה ראשונית בין YAML ו-Markdown, שני הפורמטים הנפוצים ביותר בהקשר של קונפיגורציה ותיעוד.

טבלה 5: השוואה ראשונית: YAML לעומת Markdown

קריטריון	YAML	Markdown
תחביר	רגיש לרווחים, היררכי	סימני פיסוק, גמיש
שימוש עיקרי	קונפיגורציה, הגדרות	תיעוד, תוכן טקסטואלי
קריאות	גבוהה למבנים מורכבים	גבוהה לטקסט חופשי
תמיכה במבני נתונים	מלאה: רשימות, מילונים, זימון	מוגבלת: קישורים, תמונות
עריכה ידנית	דורשת דקדקנות בהזחות	אינטואיטיבית וגמישה
המרה אחרים	לפורמטים JSON, קונפיגים	HTML, PDF, DOCX

11 השוואה מפורטת בין שלושת הפורמטים

11.1 טבלת השוואה כוללת

טבלה 6 מציגה השוואה מקיפה בין JSON, YAML, ו-Markdown, תוך בחינת היבטים טכניים ושימושיים.

טבלה 6: השוואה מפורטת: Markdown, JSON, YAML

תכונה	YAML	JSON	Markdown
תחביר	רגיש לרווחים, הזחות	סוגריים מסולסלים, פסיקים	סימני פשוטים
קריאות אנושית	גבוהה מאוד	בינונית-גבוהה	גבוהה מאוד
שימושים עיקריים	קונפיגורציה, CI/CD	העברת נתונים, API	תיעוד, בלוגים, ויקי,
תמיכה בהערות	כן: #	לא (רשמית)	כן: HTML
תמיכה במבני נתונים	רשימות, מילונים, זימון	רשימות, אובייקטים, מערכים	מוגבלת למאוד
קפדנות סינטקס	בינונית (רגישות לרווחים)	גבוהה (קפדנית מאוד)	נמוכה (גמישות)
גודל קובץ	קומפקטי (פחות תווים)	מילולי (יותר תווים)	קומפקטי
תמיכה בשפות תכנות	רחבה (PyYAML, js-yaml)	אוניברסלית (כל שפה)	רחבה (parsers)
עיבוד מכונה	בינוני (parser דורש)	מהיר וקל	לא מתאים
בקרת טעויות	טעויות נפוצות הזחה	קל לזהות (פרסרים טובים)	סובלנית (לא שוברת)
יתרונות	קריא, ידידותי, תומך הערות	אוניברסלי, מהיר, מבני	פשוט, אינטואיטיבי, גמיש
חסרונות	רגיש לרווחים, איטי לפרסר	ארוך, ללא הערות, קפדני	לא מבני, לא לנתונים

12 דוגמאות קוד מעשיות

בפרק זה נציג דוגמאות קוד זהות בשלושת הפורמטים, המדגימות את ההבדלים התחביריים והשימושיים. הדוגמאות מבוססות על הגדרת סוכן AI עם מטא-דאטה, קונפיגורציה, וכלים זמינים.

12.1 דוגמה ב-YAML

פורמט YAML נפוץ מאוד בהגדרות קונפיגורציה של סוכנים ב-Claude CLI. להלן דוגמה של קובץ הגדרת סוכן:

YAML Agent Configuration Example

```
1 # Agent Configuration
2 agent:
3   name: Expert Analyst
4   version: 1.0.0
5   description: An expert analyst agent for structured reports
6
7 config:
8   model: claude-sonnet-4
9   temperature: 0.7
10  max_tokens: 4000
11
12 tools:
13   - name: WebSearch
14     enabled: true
15   - name: Read
16     enabled: true
17   - name: Write
18     enabled: true
19
20 prompt: |
21   You are an expert analyst specializing in
22   structured technical reports.
```

12.2 דוגמה ב-JSON

פורמט JSON נפוץ בהעברת נתונים בין מערכות ובתגובות API. להלן אותה הגדרה בפורמט JSON:

JSON Agent Configuration Example

```
1 {
2   "agent": {
3     "name": "Expert Analyst",
4     "version": "1.0.0",
5     "description": "An expert analyst agent for structured
reports"
6   },
7   "config": {
8     "model": "claude-sonnet-4",
9     "temperature": 0.7,
10    "max_tokens": 4000
11  },
12  "tools": [
13    {
14      "name": "WebSearch",
15      "enabled": true
16    },
17    {
18      "name": "Read",
19      "enabled": true
20    },
21    {
22      "name": "Write",
23      "enabled": true
24    }
25  ],
26  "prompt": "You are an expert analyst specializing in structured
technical reports."
27 }
```

12.3 דוגמה ב-Markdown

פורמט Markdown אינו מתאים להגדרת קונפיגורציה מובנית, אך משמש לתיעוד ומדריכים. להלן תיאור טקסטואלי של אותו סוכן:

Markdown Agent Configuration Example

```
1 # Agent Configuration: Expert Analyst
2
3 **Version:** 1.0.0
4
5 **Description:** An expert analyst agent for structured technical
6 reports
7
8 ## Configuration
9
10 - **Model:** claude-sonnet-4
11 - **Temperature:** 0.7
12 - **Max Tokens:** 4000
13
14 ## Available Tools
15
16 - **WebSearch** (enabled)
17 - **Read** (enabled)
18 - **Write** (enabled)
19
20 ## System Prompt
21
22 You are an expert analyst specializing in structured technical
23 reports.
```

13 השוואה כללית והמלצות שימוש

13.1 מתי להשתמש בכל פורמט?

טבלה 7 מסכמת את המצבים המומלצים לשימוש בכל אחד מהפורמטים:

טבלה 7: המלצות שימוש לפי מקרה שימוש

מקרה שימוש	פורמט מומלץ
קובצי קונפיגורציה לסוכנים	YAML - קריא, תומך הערות, ידידותי למשתמש
העברת נתונים בין מערכות	JSON - אוניברסלי, מהיר לעיבוד, תמיכה רחבה
תיעוד טכני ומדריכים	Markdown - קל לקריאה, קל לכתיבה, גמיש
תגובות API	JSON - תקן בתעשייה, עיבוד מכונה אופטימלי
קובצי CI/CD	YAML - תמיכה מובנית ב-GitHub Actions, GitLab CI
מסמכים הניתנים לעריכה	Markdown - עריכה פשוטה, המרה ל-HTML/PDF
אחסון נתונים מובנים	JSON - תקן נתונים, תאימות גבוהה
קובצי הגדרות ידידותיים	YAML - פחות מילוליות, קריאות גבוהה

13.2 שילוב הפורמטים בפרויקט Claude CLI

בפרויקט בניית סוכני AI באמצעות Claude CLI, כל אחד מהפורמטים ממלא תפקיד חשוב:

YAML: משמש להגדרת סוכנים ב-`claude/agents/*.yaml`, לקובצי קונפיגורציה של `skills`, ולהגדרות CI/CD. יתרון מרכזי: אפשר להוסיף הערות מפורטות להסברת הגדרות מורכבות.

- **JSON**: משמש בתגובות API מ-Claude, בקובצי מצב (status.json), ובהעברת נתונים בין מערכות. יתרון מרכזי: עיבוד מהיר ותמיכה אוניברסלית.

- **Markdown**: משמש לתיעוד (README.md), למדריכי skills (SKILL.md), ולדוחות איכות (reviews/*.md). יתרון מרכזי: קל לקריאה וכתיבה, ניתן להמיר ל-HTML או PDF.

13.3 סיכום ומסקנות

כל אחד משלושת הפורמטים – JSON, YAML, ו-Markdown – נועד לצרכים שונים ומשלים את האחרים:

1. **YAML** הוא הבחירה האידיאלית לקובצי קונפיגורציה הדורשים קריאות גבוהה ויכולת להוסיף הערות הסבר. הוא משמש כתקן בכלי CI/CD מודרניים וב-Claude CLI.

2. **JSON** הוא תקן אוניברסלי להעברת נתונים ותגובות API, עם תמיכה רחבה בכל שפות התכנות. הוא אידיאלי למצבים הדורשים עיבוד מכונה מהיר ואמין.

3. **Markdown** הוא הפשוט ביותר לכתיבה ולקריאה, אך אינו מתאים להגדרת מבני נתונים מורכבים. תפקידו העיקרי הוא תיעוד טכני ומדריכי משתמש.

בפיתוח סוכני AI מודרניים, חשוב לדעת מתי להשתמש בכל פורמט כדי למקסם את היעילות, הקריאות, והתחזוקה של המערכת.