

## פרק 2

# אקוסיסטם הבינה המלאכותית -- מפת הכלים למנהל המודרני

## מטרות הלמידה

- הכרת כל רכיבי האקוסיסטם: מודלי שפה, ספריות, מוטמעים, בסיסי נתונים
- הבנת היחסים והתלויות בין הרכיבים השונים
- יכולת לקבל החלטות מושכלות על בחירת טכנולוגיות
- הערכת עלויות וביצועים של פתרונות שונים

## 1.2 פתח דבר: מסע בג'ונגל הטכנולוגי

דמיינו את עצמכם עומדים בפתחו של יער עצום. מכל עבר צצים שמות זרים: LangChain, OpenAI, ChromaDB, Pinecone. כל כלי מבטיח פלאים, כל ספק מציע את הפתרון המושלם. עבור מנהל שרוצה להטמיע בינה מלאכותית בארגון שלו, הבחירה הנכונה יכולה להיות מכרעת -- ההבדל בין הצלחה מסחררת לבין כישלון יקר.

בפרק הקודם למדנו מהם מודלי שפה גדולים ומה הכוח שלהם. אבל מודל שפה לבדו, כמו מנוע בלי מכונית, אינו מספיק כדי לנסוע. צריך אקוסיסטם שלם: ספריות שמחברות בין רכיבים, מסדי נתונים שמאחסנים זיכרון, כלים שמנהלים תהליכים אוטומטיים, ופלטפורמות שמאפשרות לכל זה לעבוד יחד בהרמוניה.

הפרק הזה הוא המפה שלכם לג'ונגל הטכנולוגי. נחלק את האקוסיסטם לרכיבים מרכזיים, נבין מה כל אחד עושה, ונלמד מתי להשתמש במה. בסוף הפרק תדעו לתכנן ארכיטקטורה שלמה, להשוות בין ספקים, ולחשב עלויות. זה לא רק ידע טכני -- זה כוח אסטרטגי.

## 2.2 שכבות האקוסיסטם: ארכיטקטורה מלמעלה למטה

כדי להבין את האקוסיסטם, נחלק אותו לחמש שכבות מרכזיות:

### 1.2.2 שכבת הליבה: מודלי השפה

זוהי השכבה הבסיסית -- המוח של המערכת. כאן יושבים מודלי השפה הגדולים עצמם. יש לנו שני מסלולים עיקריים:

**ספקי ענן (Cloud Providers):** אלו חברות שמריצות מודלים ענקיים על תשתיות ענן ומאפשרות לנו לגשת אליהם דרך API. המובילים:

□ **OpenAI** -- החלוץ והמוביל. GPT-4, GPT-4 Turbo, ו-GPT-3.5 הם הסטנדרט התעשייתי. יתרון מרכזי: בשלות, תיעוד עשיר, ואקוסיסטם תומך ענק. חיסרון: עלות גבוהה יחסית, תלות בספק בודד.

□ **Anthropic** -- היריבה המתקדמת. מודל Claude 3.5 Sonnet ו-Claude Opus 4.5 מציעים חלון הקשר ענק (עד 000,002 טוקנים), דיוק גבוה במשימות מורכבות, ודגש על בטיחות. אידיאלי למשימות הדורשות הבנת הקשר עמוקה.

□ **Google** -- Gemini Pro ו-Gemini Ultra משלבים מולטימודליות מתקדמת (טקסט, תמונה, וידאו). יתרון משמעותי: אינטגרציה חלקה עם Google Workspace.

□ **Meta** -- Llama 3 (בגרסאות B8, B7, B504) הוא מודל קוד פתוח. למה זה משמעותי? תוכלו להוריד ולהריץ אותו על השרתים שלכם. אין תלות בספק חיצוני, אין דליפת מידע רגיש החוצה.

□ **DeepSeek** -- השחקן הסיני המפתיע. DeepSeek-V3 מציע ביצועים מרשימים במחיר נמוך משמעותית. בעיקר למשימות טכניות וקוד.

**מודלים מקומיים (Self-Hosted):** כאן אנחנו מורידים את המודל ומריצים אותו על החומרה שלנו:

□ **Llama 3.1 (8B/70B)** -- מודל קוד פתוח מצוין לריצה מקומית. גרסת B8 רצה אפילו על לפטופ חזק, B7 דורש שרת עם UPG.

□ **Mistral 7B** -- קטן, מהיר, יעיל. מצוין לסטארטאפים שרוצים פתרון זול ומקומי.

□ **Qwen 2.5** -- מודל סיני מתקדם, מצוין לתמיכה רב-לשונית.

למה לבחור ב-Self-Hosted? שלוש סיבות עיקריות:

1. **פרטיות מוחלטת** -- נתונים רגישים לא עוזבים את הארגון.

2. **עלות צפויה** -- שילמתם על החומרה פעם אחת, אין הפתעות בחשבון.

3. **התאמה אישית** -- אפשר לעשות Fine-tuning ייעודי.

## 2.2.2 שכבת החיבור: OpenRouter -- הרכזת של מודלים

OpenRouter הוא כמו שדה תעופה שמחבר אתכם לכל היעדים. במקום לפתוח חשבון בנפרד אצל OpenAI, Anthropic, ו-Google, אתם נרשמים פעם אחת ל-OpenRouter ומקבלים גישה למעל 100 מודלים שונים דרך API אחיד. **יתרונות מנהליים:**

□ גמישות -- החלפת מודל זה שורת קוד אחת

□ השוואת עלויות -- תוכלו לנסות מודלים שונים בלי להתחייב

□ גיבוי אוטומטי -- אם ספק אחד נופל, OpenRouter מעביר למודל חלופי

**דוגמה מעשית:** חברת תמיכה טכנית השתמשה ב-GPT-4 לניתוח פניות מורכבות. אבל 70% מהפניות היו פשוטות, ו-GPT-4 יקר מדי בשבילן. עם OpenRouter הם יישמו לוגיקה: פניות פשוטות ל-GPT-3.5 (זול), פניות מורכבות ל-Claude Sonnet (מדויק). חיסכו 60% בעלויות בלי לוותר על איכות.

### 3.2.2 שכבת הפיתוח: ספריות אינטגרציה

כדי לבנות מערכת אמיתית, לא מספיק לשלוח בקשה ל-API ולקבל תשובה. צריך לנהל שיחות, לזכור הקשר, לחבר בסיסי נתונים, לטפל בשגיאות. כאן נכנסות הספריות:

#### LangChain -- הסוכן המקצועי

LangChain היא הספרייה הפופולרית ביותר לבניית אפליקציות LLM. היא מציעה:

□ **Chains** -- שרשרת פעולות. לדוגמה: קח שאלה → חפש במסד נתונים → שלח ל-LLM → עצב תשובה.

□ **Agents** -- סוכנים אוטונומיים שיודעים לבחור כלים. "איזה טיסות זולות לברלין?" → הסוכן מבין שצריך לקרוא ל-API של טיסות, מנתח תוצאות, ומחזיר תשובה.

□ **Memory** -- זיכרון שיחה. ה-LLM זוכר מה דיברתם לפני 10 הודעות.

□ **Retrievers** -- חיבור לבסיסי נתונים וקטוריים (נדבר עליהם בהמשך).

**מתי להשתמש ב-LangChain?** כשאתם בונים משהו מורכב -- סוכן שירות, מערכת RAG, אוטומציה רב-שלבית.

#### LangGraph -- תזמור תהליכים

LangGraph הוא ההמשך של LangChain, ממוקד בניהול תהליכים מורכבים עם מעברים ותנאים. דמיינו תהליך אישור הזמנה:

1. בדיקת מלאי

2. אישור מנהל (אם מעל 10,000 ש"ח)

3. שליחה ללוגיסטיקה

4. עדכון לקוח

LangGraph מאפשר לכם לתכנן את התהליך כגרף זרימה, עם צמתים (פעולות) וקשתות (תנאים). כל צומת יכול להיות LLM, שאילתת מסד נתונים, או קריאה חיצונית.

#### Pydantic AI -- המובנה והמסודר

אם LangChain הוא הסוכן הגמיש, Pydantic AI הוא הבנקאי המדויק. הספרייה מתמחה במבנה ובולידציה:

□ הגדרת מבני נתונים נוקשים

□ ולידציה אוטומטית של תשובות LLM

□ אכיפת פורמטים (JSON Schema)

**דוגמה:** אתם רוצים ש-LLM יחלץ מפניית לקוח: שם, מייל, נושא, רמת דחיפות. Pydantic AI מגדיר מבנה קפדני, ואם ה-LLM מחזיר משהו שלא תואם -- יש שגיאה מיידית.

## 4.2.2 שכבת ההטמעה: Embeddings ומסדי נתונים וקטוריים

אחת התובנות המרכזיות בעולם ה-AI המודרני היא שמילים הן לא רק תווים -- יש להן משמעות גיאומטרית. טכנולוגיית Embeddings הופכת טקסט למספרים (וקטורים), כך שמחשב יכול "להבין" דמיון סמנטי.

### מהם Embeddings?

תארו לעצמכם מרחב של מאות או אלפי ממדים. כל מילה או משפט הוא נקודה במרחב הזה. משפטים דומים במשמעות קרובים גיאומטרית; משפטים שונים רחוקים. לדוגמה:

□ " כלב" ו-"גור" -- קרובים מאוד

□ "כלב" ו-"מכונית" -- רחוקים

**למה זה חשוב?** כי ככה בונים חיפוש סמנטי. במקום לחפש מילת מפתח מדויקת (כמו Google פעם), אפשר לחפש לפי כוונה.

**שאלה:** "איך מגישים תביעת ביטוח?" **מסמך במערכת:** "הליך הגשת דרישה לפיצוי" חיפוש רגיל לא ימצא את זה -- אין מילים משותפות. חיפוש וקטורי כן.

### מודלי Embedding מובילים

□ **OpenAI Text-Embedding-3** -- הסטנדרט. קל לשימוש, איכות מצוינת. גרסת small (זולה) וגרסת large (מדויקת).

□ **NV-Embed-v2** -- מודל מתקדם מבית NVIDIA. מצוין לטקסטים טכניים ומדעיים.

□ **BGE-M3** -- מודל סיני רב-לשוני. תומך במעל 100 שפות, כולל עברית. קוד פתוח.

### מסדי נתונים וקטוריים

אחרי שהפכנו טקסט לוקטורים, איפה נאחסן אותם? מסד נתונים רגיל (MySQL, PostgreSQL) לא יודע לעבוד עם חיפוש וקטורי. צריך מסד נתונים וקטורי (Vector Database).

#### Pinecone -- הענן המנוהל:

□ **יתרונות:** לא צריך להתקין כלום. שירות ענן מנוהל, סקיילבילי, מהיר.

□ **חסרונות:** עלות חודשית, תלות בספק.

□ **מתי להשתמש:** כשאתם רוצים לעלות מהר, בלי להתעסק בתשתיות.

#### Chroma -- הפתרון המקומי:

□ **יתרונות:** קוד פתוח, חינומי, רץ על השרת שלכם.

□ **חסרונות:** אתם צריכים לנהל: גיבויים, ביצועים, סקייל.

□ **מתי להשתמש:** כשאתם בשלב POC, או כשאתם רוצים שליטה מלאה.

#### Weaviate -- הכלי ההיברידי:

□ **יתרונות:** תומך גם בחיפוש וקטורי וגם בחיפוש טקסט רגיל. אינטגרציות עשירות.

□ **חסרונות:** מורכב יותר להקמה.

□ **מתי להשתמש:** כשאתם צריכים גמישות מקסימלית.

**Qdrant -- המהיר:**

□ **יתרונות:** ביצועים מצוינים, נכתב ב-Rust (מהיר ויעיל).

□ **חסרונות:** קהילה קטנה יותר.

□ **מתי להשתמש:** כשמהירות היא קריטית.

## 5.2.2 שכבת האוטומציה: כלים אגנטיים

השכבה העליונה היא שכבת התזמור -- הכלים שגורמים לכל המערכת לעבוד יחד ולהריץ תהליכים אוטומטיים.

**LangGraph (שוב, אבל בהקשר אחר)**

כבר דיברנו עליו כספרייה, אבל הוא גם כלי תזמור. LangGraph מאפשר לכם לבנות תהליכים רב-שלביים שבהם סוכנים שונים מתקשרים זה עם זה, מעבירים מידע, ומקבלים החלטות.

**AutoGen -- צוותי סוכנים**

AutoGen (מבית Microsoft) הוא פריימוורק לבניית מערכות רב-סוכן. דמיינו שאתם בונים מערכת לניהול פרויקטים:

□ **סוכן תכנון** -- מנתח דרישות ובונה תוכנית עבודה

□ **סוכן ביצוע** -- מקצה משימות לאנשי צוות

□ **סוכן בקרה** -- עוקב אחרי התקדמות ומתריע על עיכובים

כל סוכן הוא LLM עם הנחיות (System Prompt) ייעודיות. AutoGen מנהל את התקשורת ביניהם.

**n8n -- אוטומציה חזותית**

n8n הוא כלי No-Code / Low-Code לאוטומציה. במקום לכתוב קוד, אתם גוררים בלוקים ומחברים אותם.

**דוגמה:**

1. כל פניית לקוח במייל →

2. LLM מנתח ומקטלג →

3. אם דחוף: שולח SMS למנהל →

4. אם רגיל: פותח כרטיס ב-Jira

הכל בלי לכתוב שורת קוד אחת.

### Zapier -- השחקן הוותיק

Zapier קיים הרבה לפני AI, אבל הוא השתלב מצוין. תומך באלפי אינטגרציות (liamG, kcalS, ecrofselaS, noitoN...). לאחרונה הוסיף תמיכה ב-OpenAI ו-Antropic.

### מתי n8n ומתי Zapier?

- Zapier -- אם אתם רוצים פשטות ותמיכה ענקית בשירותים חיצוניים
- n8n -- אם אתם רוצים שליטה, תמחור טוב יותר, ואפשרות ל-Self-Hosted

## 3.2 מתי להשתמש במה: מטריצת החלטות

עכשיו שאנחנו מכירים את כל השחקנים, איך בוחרים? הנה מטריצה מנהלית:

### 1.3.2 בחירת ספק LLM

טבלה 1.2: מטריצת בחירת מודל שפה

קריטריון	בחירה	הסבר
דיוק גבוה, משימות מורכבות	Claude Opus / GPT-4	הטובים ביותר לחשיבה מורכבת
עלות נמוכה, נפח גבוה	GPT-3.5 / DeepSeek	יחס מחיר-ביצועים מעולה
פרטיות, נתונים רגישים	Llama 3 (Self-Hosted)	שום דבר לא עוזב את הארגון
מולטימודליות	Gemini Pro / GPT-4 Vision	תמיכה בתמונות ווידאו
חלון הקשר ענק	Claude 3.5 Sonnet	עד 200K טוקנים

### 2.3.2 בחירת מסד נתונים וקטורי

### 3.3.2 החלטת ענן מול On-Premise

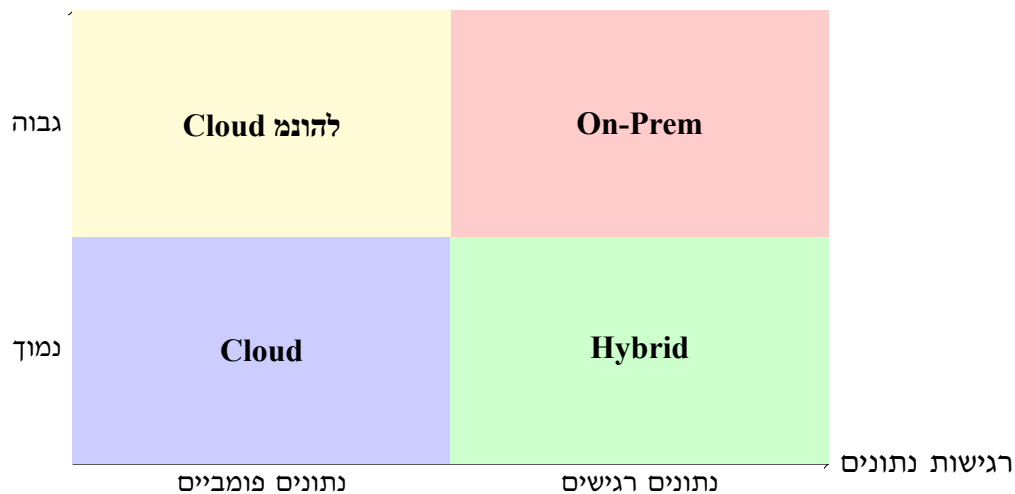
פענוח המטריצה:

- רביע שמאלי תחתון (כחול): נתונים לא רגישים, נפח נמוך → Cloud (OpenAI, Anthropic)
- רביע ימני תחתון (ירוק): נתונים רגישים, נפח נמוך → Hybrid (חלק בענן, חלק מקומי)
- רביע ימני עליון (אדום): נתונים רגישים, נפח גבוה → On-Prem (Llama 3) על שרתים (פנימיים)
- רביע שמאלי עליון (צהוב): נתונים לא רגישים, נפח גבוה → Cloud מנוהל עם הסכם ארגוני

טבלה 2.2: מטריצת בחירת מסד נתונים וקטורי

הסבר	בחירה	תרחיש
חינמי, פשוט, מקומי	Chroma	POC / אב טיפוס
מנוהל, אמין, לא צריך DevOps	Pinecone	ייצור, סטארטאפ
שליטה מלאה, גמישות	Weaviate ו Qdrant	ארגון גדול, On-Prem
אופטימיזציה קיצונית	Qdrant	צורך במהירות מקסימלית

נפח שימוש חודשי



איור 1.2: מטריצת החלטה: ענן מול On-Premise

## 4.2 נוסחאות מנהליות: חשבון כלכלי

כמנהלים, אנחנו צריכים להצדיק כל החלטה טכנולוגית בשפה כלכלית. הנה שתי נוסחאות קריטיות:

### 1.4.2 נוסחת TCO -- עלות בעלות כוללת

$$(2.1) \quad TCO = C_{\text{license}} + C_{\text{infra}} + (C_{\text{HR}} \times 12)$$

פירוק הנוסחה:

$C_{\text{license}}$  -- עלות רישוי שנתית (מנויים ל-IPA, רישיונות תוכנה) □

$C_{\text{infra}}$  -- עלות תשתיות (שרתים, אחסון, רשת, חשמל) □

$C_{\text{HR}}$  -- עלות כוח אדם חודשית (מפתחים, DevOps, מנהלי מערכת) □

דוגמה 1: פתרון ענן טהור (OpenAI)

□ רישוי: \$5,000 לחודש (\$60,000 לשנה)

□ תשתיות: \$0 (ספק מנוהל)

□ כוח אדם: מפתח חצי משרה (\$4,000 לחודש)

$$(2.2) \quad TCO = 60,000 + 0 + (4,000 \times 12) = \$108,000$$

### דוגמה 2: פתרון On-Premise (Llama 3)

□ רישוי: \$0 (קוד פתוח)

□ תשתיות: שרת עם 8 GPUs (\$80,000 לשנה, פחת על 5 שנים = \$16,000 לשנה) + חשמל וקירור (\$12,000 לשנה)

□ כוח אדם: מפתח + DevOps (\$10,000 לחודש)

$$(2.3) \quad TCO = 0 + 28,000 + (10,000 \times 12) = \$148,000$$

**מסקנה:** בטווח הקצר, הענן זול יותר. אבל אם נפח השימוש גדל -- On-Prem משתלם יותר.

## 2.4.2 Latency -- זמן תגובה

$$(2.4) \quad T_{\text{total}} = T_{\text{network}} + T_{\text{processing}} + T_{\text{model}}$$

**פירוק הנוסחה:**

□  $T_{\text{network}}$  -- זמן העברת הבקשה והתשובה ברשת (TTR)

□  $T_{\text{processing}}$  -- זמן עיבוד מקדים (gniddebme, חיפוש במסד נתונים)

□  $T_{\text{model}}$  -- זמן ההסקה של המודל עצמו

### דוגמה: שאלת RAG

□  $T_{\text{network}} = 50\text{ms}$  (לענן בחו"ל) או  $2\text{ms}$  (לשרת מקומי)

□  $T_{\text{processing}} = 100\text{ms}$  (Embedding + חיפוש ב-Pinecone)

□  $T_{\text{model}} = 800\text{ms}$  (GPT-4)

**סה"כ (ענן):**

$$(2.5) \quad T_{\text{total}} = 50 + 100 + 800 = 950\text{ms}$$

**סה"כ (מקומי):**

$$(2.6) \quad T_{\text{total}} = 2 + 100 + 800 = 902\text{ms}$$

**מסקנה:** זמן רשת נראה קטן, אבל בנפח גבוה (אלפי בקשות ביום) -- הוא משמעותי.



טבלה 3.2: השוואת מחירי מודלים מרכזיים

מודל	קלט M1/(\$) (טוקנים)	פלט M1/(\$) (טוקנים)	משוקלל*
GPT-4 Turbo	\$10.00	\$30.00	\$20.00
GPT-3.5 Turbo	\$0.50	\$1.50	\$1.00
Claude 3.5 Sonnet	\$3.00	\$15.00	\$9.00
Claude 3 Haiku	\$0.25	\$1.25	\$0.75
Gemini Pro	\$0.50	\$1.50	\$1.00
DeepSeek-V3	\$0.27	\$1.10	\$0.69
Llama 3.1 70B**	\$0.00	\$0.00	\$0.00***

\* משוקלל: הנחה של 50% קלט, 50% פלט  
 \*\* דרך OpenRouter או ספקים אחרים  
 \*\*\* עלות אפסית לשימוש, אבל יש עלות תשתית

## 5.2 השוואת מחירים: מי הכי משתלם?

מחירי API משתנים כל הזמן, אבל הנה תמונת מצב (נכונה לדצמבר 2024):  
**תובנות מהטבלה:**

1. GPT-4 הוא הכי יקר -- פי 20 מ-GPT-3.5. האם התוצאה מצדיקה? תלוי במשימה.
2. Claude Haiku ו-DeepSeek -- אלטרנטיבות זולות ומצוינות לנפח גבוה.
3. Llama 3 -- עלות אפסית לשימוש, אבל צריך להשקיע בתשתית.

## 6.2 דוגמאות מעשיות מהשטח

### 1.6.2 דוגמה 1: סטארטאפ בשלב Seed -- בחירת Stack

**תרחיש:** חברת FinTech עם 5 עובדים, רוצה לבנות עוזר פיננסי אישי. תקציב: \$2,000 לחודש.  
**צרכים:**

- עיבוד שאלות פיננסיות פשוטות (80%) ומורכבות (20%)
- אחסון ידע על מוצרים פיננסיים (200 מסמכים)

□ אינטגרציה עם בנקים (API)

#### ארכיטקטורה מומלצת:

□ **LLM**: OpenRouter עם GPT-3.5 לשאלות פשוטות, Claude Sonnet למורכבות

□ **Embeddings**: OpenAI Text-Embedding-3-small (זול)

□ **Vector DB**: Pinecone תוכנית חינמית (עד 100K וקטורים)

□ **Framework**: LangChain (קהילה גדולה, הרבה דוגמאות)

□ **Automation**: n8n (Self-Hosted, חינמי)

#### עלויות:

□ OpenRouter: \$800/חודש (בממוצע)

□ Pinecone: \$0 (תוכנית חינמית)

□ n8n: \$0 (Self-Hosted על AWS EC2 קטן, \$20/חודש)

□ פיתוח: מפתח אחד חצי משרה

□ סה"כ: \$820/חודש -- מתחת לתקציב, עם מרווח לגדילה.

## 2.6.2 דוגמה 2: ארגון גדול -- מעבר מ-ChatGPT לפתרון ארגוני

**תרחיש:** חברת ייעוץ עם 500 עובדים. כרגע כולם משתמשים ב-ChatGPT אישי. הבעיה: אין שליטה, נתונים דולפים, אין התאמה אישית.

#### דרישות:

□ גישה למידע פנימי (מדיניות, פרויקטים, לקוחות)

□ פרטיות מלאה -- נתונים לא יוצאים

□ יכולת ביקורת -- מי שאל מה ומתי

□ התאמה אישית לתהליכים של הארגון

#### ארכיטקטורה מומלצת:

□ **LLM**: Azure OpenAI (גרסה ארגונית -- נתונים לא משמשים לאימון)

□ **RAG**: Weaviate על שרתים פנימיים

□ **Embeddings**: Azure OpenAI Embeddings

□ **Framework**: LangChain עם LangSmith (ניטור ולוגים)

□ **UI**: פורטל פנימי מבוסס Streamlit

#### יתרונות:

□ נתונים נשארים ב-Tenant הארגוני של Azure

□ אפשר לחבר לכל מסדי הנתונים הפנימיים

□ לוגים מלאים לביקורת

□ התאמה אישית -- אפשר לכוון את המודל לתהליכים ספציפיים

#### עלויות (הערכה):

□ Azure OpenAI: \$15,000/חודש (500 משתמשים)

□ Weaviate על Azure VM: \$2,000/חודש

□ פיתוח ותחזוקה: 2 מפתחים (\$20,000/חודש)

**סה"כ: \$37,000/חודש** -- נשמע יקר? לא בהשוואה ל-500 רישיונות ChatGPT Plus (\$10,000/חודש) ללא שליטה וללא התאמה אישית.

### 3.6.2 דוגמה 3: החלטת Build vs Buy

**תרחיש:** חברת e-commerce רוצה סוכן שירות אוטומטי. השאלה: לקנות פתרון מוכן (כמו Intercom) או לבנות בעצמנו?

#### ניתוח Build:

□ **עלות פיתוח:** 3 חודשי עבודה של מפתח (\$30,000)

□ **עלות ריצה:** API + שרתים (\$1,500/חודש)

□ **תחזוקה:** רבע משרה (\$3,000/חודש)

□ **סה"כ שנה ראשונה:**  $\$84,000 = \$30,000 + (\$4,500 \times 12)$

#### ניתוח Buy (Intercom):

□ **רישוי:** \$5,000/חודש (500 שיחות ביום)

□ **אינטגרציה:** חודש עבודה (\$10,000)

□ **תחזוקה:** כמעט אפסית

□ **סה"כ שנה ראשונה:**  $\$70,000 = \$10,000 + (\$5,000 \times 12)$

**החלטה:** בשנה הראשונה, Buy זול יותר. אבל:

□ אם הנפח גדל (1,000 שיחות ביום) -- Intercom יעלה ל-\$10,000/חודש

□ פתרון Build נשאר באותה עלות (או עולה מעט)

□ פתרון Build מאפשר התאמה מלאה לתהליכים

**המלצה:** התחילו עם Buy (מהיר, מוכח). אחרי שנה, אם הנפח גדל -- עברו ל-Build.

## 7.2 תרגילים

### 1.7.2 תרגיל 1 (תיאורטי): בניית ארכיטקטורה לפרויקט AI

**תרחיש:** אתם סמנכ"ל טכנולוגיות בחברת ביטוח בינונית (200 עובדים). מנכ"ל מבקש מכם לבנות מערכת AI שתענה על שאלות סוכני הביטוח על מדיניות ותקנות (5,000 מסמכים פנימיים).

#### דרישות:

- פרטיות מלאה -- מסמכים רגישים
- זמן תגובה: עד 3 שניות
- תקציב: \$10,000/חודש
- 50 שאלות ביום במוצע

#### משימה:

1. בחרו ספק LLM (ענן / מקומי / היברידי)
2. בחרו מודל Embedding
3. בחרו מסד נתונים וקטורי
4. בחרו ספריית אינטגרציה
5. הצדיקו כל בחירה
6. חשבו TCO לשנה

### 2.7.2 תרגיל 2 (תיאורטי): חישוב TCO לשלושה תרחישים

השוו TCO לשלוש אפשרויות:

#### תרחיש א': ענן טהור

□ GPT-4 דרך OpenAI

□ Pinecone מנוהל

□ מפתח חצי משרה

#### תרחיש ב': היברידי

□ Azure OpenAI (ארגוני)

□ Weaviate על Azure VM

□ מפתח + מנהל מערכת (שני חצאי משרה)

#### תרחיש ג': On-Premise מלא

□ Llama 3.1 70B על שרת פנימי

□ Chroma מקומי

□ מפתח + DevOps + מנהל מערכת

חשבו TCO לשנה לכל תרחיש. איזה משתלם לנפח של:

- 1,000 שאלות ביום?
- 10,000 שאלות ביום?
- 100,000 שאלות ביום?

### 3.7.2 תרגיל 3 (תיאורטי): ניתוח Trade-offs בין Pinecone ל-Chroma

מנהל פיתוח שואל אתכם: "למה לא פשוט להשתמש ב-Chroma? זה בחינם!"  
**משימה:**

1. רשמו 5 יתרונות של Pinecone
2. רשמו 5 יתרונות של Chroma
3. בנו טבלת החלטה: באיזה מקרים כל אחד עדיף
4. הסבירו למה "בחינם" לא תמיד זול יותר

### 4.7.2 תרגיל 4 (תיאורטי): תכנון אסטרטגיית Vendor

אתם אחראים על אסטרטגיית AI ארוכת טווח. מנכ"ל דואג מ-Vendor Lock-in.  
**משימה:**

1. הסבירו מהם הסיכונים של תלות בספק בודד
2. תכננו אסטרטגיית Multi-Vendor (יותר מספק אחד)
3. רשמו 3 טכניקות למניעת Lock-in
4. נתחו: האם OpenRouter פותר את הבעיה? למה כן / למה לא?

### 5.7.2 תרגיל 5 (תיאורטי): בניית RFP לבחירת ספק AI

**תרחיש:** הארגון שלכם רוצה לבחור ספק AI לטווח ארוך. עליכם לכתוב RFP (Request for Proposal).  
**משימה:** כתבו RFP שכולל:

1. רקע על הארגון וצרכיו
2. דרישות פונקציונליות (מה המערכת צריכה לעשות)
3. דרישות לא-פונקציונליות (ביצועים, אבטחה, זמינות)
4. קריטריוני הערכה (איך תבחרו בין הצעות)
5. לוח זמנים

**6.7.2 תרגיל 6 (קוד Python): השוואת מחירי API בין ספקים**

כתבו סקריפט Python שמקבל:

□ מספר טוקנים קלט

□ מספר טוקנים פלט

□ מספר בקשות לחודש

ומחשב עלות חודשית עבור:

□ GPT-4 Turbo

□ GPT-3.5 Turbo

□ Claude 3.5 Sonnet

□ Gemini Pro

**בנוסף:** הציגו את התוצאות בגרף עמודות.  
**קוד התחלתי:**

## השוואת מחירי IPA

```

1 # API price comparison between providers
2
3 # Prices ($/1M tokens) - update with actual prices
4 PRICING = {
5     "GPT-4 Turbo": {"input": 10.0, "output": 30.0},
6     "GPT-3.5 Turbo": {"input": 0.5, "output": 1.5},
7     "Claude 3.5 Sonnet": {"input": 3.0, "output": 15.0},
8     "Gemini Pro": {"input": 0.5, "output": 1.5},
9 }
10
11 def calculate_monthly_cost(model_name, input_tokens,
12                             output_tokens, requests_per_month):
13     """
14     Calculate monthly cost for a given model
15
16     Args:
17         model_name: Model name
18         input_tokens: Input tokens per request
19         output_tokens: Output tokens per request
20         requests_per_month: Requests per month
21
22     Returns:
23         Monthly cost in dollars
24     """
25     pricing = PRICING[model_name]
26
27     # Calculate total tokens per month
28     total_input = input_tokens * requests_per_month
29     total_output = output_tokens * requests_per_month
30
31     # Calculate cost (price per million, divide by million)
32     input_cost = (total_input / 1_000_000) * pricing["input"]
33     output_cost = (total_output / 1_000_000) * pricing["output"]
34
35     return input_cost + output_cost
36
37 # Usage example
38 if __name__ == "__main__":
39     # Example parameters
40     input_tokens = 500
41     output_tokens = 200
42     requests = 10_000 # 10K requests per month
43
44     print("Monthly cost comparison:")
45     print(f"Parameters: {input_tokens} input tokens, "
46           f"{output_tokens} output tokens, {requests:,} requests\n")
47
48     for model in PRICING.keys():
49         cost = calculate_monthly_cost(model, input_tokens,
50                                     output_tokens, requests)
51         print(f"{model:20s}: ${cost:8.2f}")
52
53     # TODO: Add graphical display with matplotlib

```

**7.7.2 תרגיל 7 (קוד Python): בדיקת Latency של מודלים**

כתבו סקריפט שבודק זמן תגובה ממוצע של מודלים שונים.

**דרישות:**

1. שלחו 10 בקשות זהות לכל מודל
2. מדדו זמן תגובה לכל בקשה
3. חשבו ממוצע, חציון, סטיית תקן
4. הציגו תוצאות בטבלה
5. הציגו Box Plot להשוואה ויזואלית

**קוד התחלתי:**



## בדיקת Latency של מודלים

```

1 import time
2 import statistics
3 from openai import OpenAI
4 import anthropic
5
6 # Configuration
7 MODELS = {
8     "gpt-3.5-turbo": "openai",
9     "gpt-4-turbo": "openai",
10    "claude-3-5-sonnet-20241022": "anthropic",
11 }
12
13 TEST_PROMPT = "What is the capital of France? Answer in one word."
14 NUM_TESTS = 10
15
16 def test_openai_latency(model_name, num_tests=10):
17     """Test latency for OpenAI model"""
18     client = OpenAI() # API key from .env
19     latencies = []
20
21     for i in range(num_tests):
22         start = time.time()
23         response = client.chat.completions.create(
24             model=model_name,
25             messages=[{"role": "user", "content": TEST_PROMPT}],
26             max_tokens=10
27         )
28         end = time.time()
29         latencies.append((end - start) * 1000) # Convert to ms
30         time.sleep(1) # Prevent rate limiting
31
32     return latencies
33
34 def test_anthropic_latency(model_name, num_tests=10):
35     """Test latency for Anthropic model"""
36     client = anthropic.Anthropic() # API key from .env
37     latencies = []
38
39     for i in range(num_tests):
40         start = time.time()
41         message = client.messages.create(
42             model=model_name,
43             max_tokens=10,
44             messages=[{"role": "user", "content": TEST_PROMPT}]
45         )
46         end = time.time()
47         latencies.append((end - start) * 1000)
48         time.sleep(1)
49
50     return latencies
51
52 def analyze_latencies(latencies, model_name):
53     """Statistical analysis of latencies"""
54     return {
55         "model": model_name,
56         "mean": statistics.mean(latencies),
57         "median": statistics.median(latencies),
58         "stdev": statistics.stdev(latencies),
59         "min": min(latencies), 71
60         "max": max(latencies)
61     }
62

```

## 8.2 סיכום: המפה שלכם לאקוסיסטם

עברנו מסע ארוך בג'ונגל הטכנולוגי. למדנו שאקוסיסטם הבינה המלאכותית אינו רק מודל שפה בודד, אלא מערכת שלמה של רכיבים מתוחכמים שעובדים יחד:

- **שכבת הליבה** -- מודלי שפה, ענן ומקומיים
  - **שכבת החיבור** -- OpenRouter כרכזת גמישה
  - **שכבת הפיתוח** -- ספריות כמו Pydantic AI, LangGraph, LangChain
  - **שכבת ההטמעה** -- Embeddings ומסדי נתונים וקטוריים
  - **שכבת האוטומציה** -- כלים אגנטיים לתזמור תהליכים
- למדנו שאין פתרון אחד מושלם. כל בחירה תלויה בהקשר:
- **ענן** -- מהירות, נוחות, אבל תלות בספק
  - **מקומי** -- פרטיות, שליטה, אבל מורכבות
  - **היברידי** -- האיזון הטוב ביותר לארגונים גדולים

למדנו לחשב TCO ו-Latency, להשוות בין ספקים, ולקבל החלטות מושכלות. הידע הזה אינו רק טכני -- הוא אסטרטגי. בעולם שבו AI הופך למרכיב עסקי קריטי, היכולת לבחור את הכלים הנכונים היא יתרון תחרותי.

בפרק הבא נצלול לעומק טכני יותר -- נלמד איך לתקשר עם מודלים השפה דרך REST APIs ו-JSON. זו השפה שבה מדברים עם מכונות, והיכולת להבין אותה היא המפתח לבניית מערכות אמיתיות.

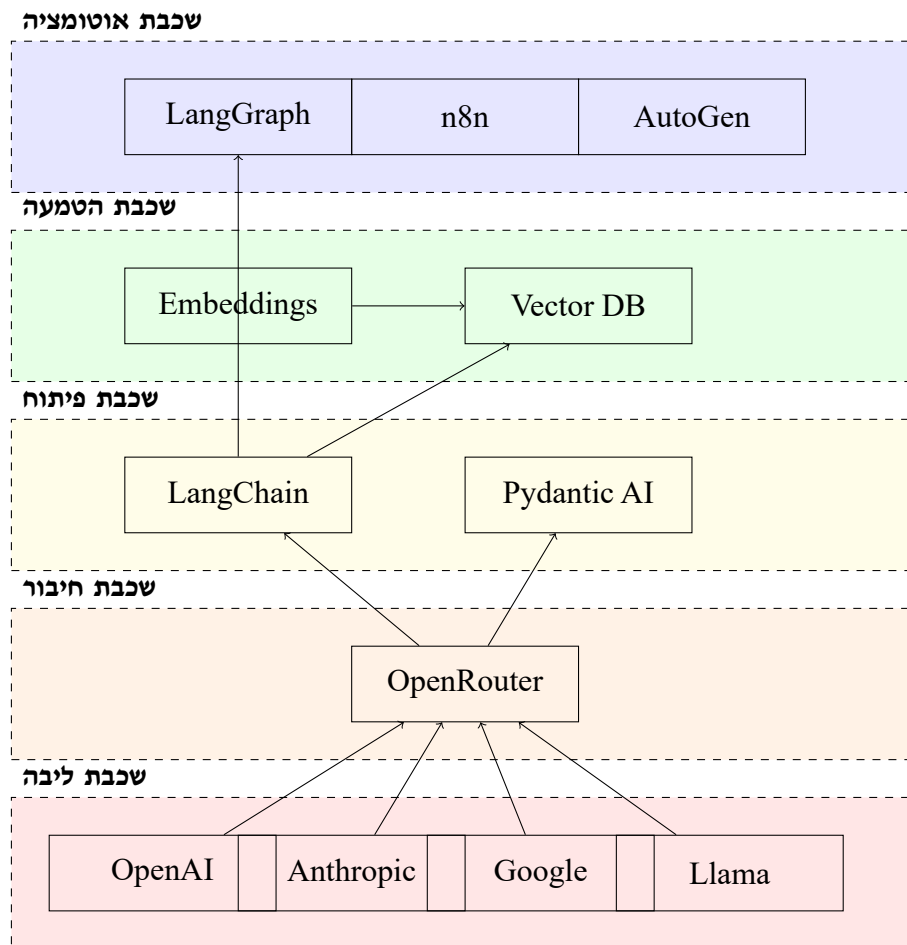
## נספח: גרפים ותרשימים

**תרשים 1: ארכיטקטורת אקוסיסטם מלאה**

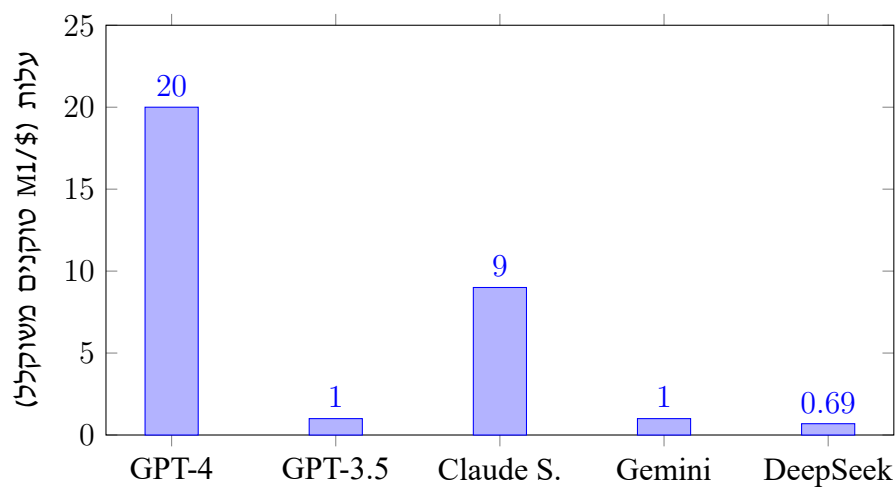
**תרשים 2: גרף עמודות -- השוואת מחירים**

**רשימת קריאה מומלצת**

- <https://python.langchain.com/> -- noitatnemucoD niahCgnaL
- <https://openrouter.ai/models> -- tsiL sledoM retuoRnepO
- <https://www.pinecone.io/learn/> -- retneC gninraeL enoceniP
- <https://docs.anthropic.com/> -- noitatnemucoD edualC ciporhtnA
- <https://llama.meta.com/> -- draC ledoM amallL



איור 2.2: ארכיטקטורת אקוסיסטם AI מלאה -- 5 שכבות



איור 3.2: השוואת מחירים ממוצעים בין מודלים מובילים