

פרק 2

אקויסיטטם הבינה המלאכותית -- מפת הכלים למנהל המודרני

מטרות הלמידה

- הכרת כל רכיבי האקויסיטטם: מודלי שפה, ספריות, מוטמעים, בסיסי נתונים
- הבנת היחסים והתלות בין הרכיבים השונים
- יכולת לקבל החלטות מושכלות על בחירת טכנולוגיות
- הערכת עליות וביטויים של פתרונות שונים

1.2 פתח דבר: מסע בג'ונגל הטכנולוגי

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

בפרק הקודם למדנו מהם מודלי שפה גדולים ומה הכוח שלהם. אבל מודל שפה לבדו, כמו מגוון מכוני, אינו מספיק כדי לנסוע. צרייך אקויסיטטם שלם: ספריות שמחברות בין רכיבים, מסדי נתונים שמאחסנים זיכרון, כלים שמנחים תהליכים אוטומטיים, ופלטפורמות שאפשרות לכל זה לעבוד יחד בהרמונייה.

הפרק הזה הוא המפה שלכם לג'ונגל הטכנולוגי. נחלק את האקויסיטטם לרכיבים מרכזיים, נבין מה כל אחד עשוּה, ונלמד متى להשתמש בהמה. בסוף הפרק תדעו לתוכן ארכיטקטורה שלמה, להשווות בין ספקיים, ולהשאוב עליות. זה לא רק ידע טכני -- זה כוח אסטרטגי.

2.2 שכבות האקויסיטטם: ארכיטקטורה מלמעלה למטה

כדי להבין את האקויסיטטם, נחלק אותו לחמש שכבות מרכזיות:

1.2.2 שכבת הליבה: מודלי השפה

זהו הLayer השכבה הבסיסית -- המוח של המערכת. כאן יושבים מודלי השפה הגדולים עצם. 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 Turbo, GPT-4, GPT-3.5 ו-GPT-3.5 Turbo הם הסטנדרט התעשייתי. יתרון מרכזי: בשנות, תיעוד עשיר, ואקויסיטים תומך ענק. חיסרונו: עלות גבוהה יחסית, תלות בספק בודד.

□ **Anthropic** -- היריבה המתקדמת. מודל Claude Opus 4.5- Claude Sonnet 3.5 מציגים חлон קשור ענק (עד 000,000 טוקנים), דיוק גבוה במשימות מורכבות, ודוגש על בטיחות. אידיאלי למשימות הדורשות הבנת הקשר עמוקה.

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

□ **Meta** -- Llama 3 (בגרסאות B07, B08, 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 המפתח. DeepSeek-V3 מציע ביצועים מרשים 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. גרסה 8B רצתה אפילו על לפטופ חזק, B07>Dורש שירות עם UPG.

□ **Mistral 7B** -- קטן, מהיר, יעיל. מצוין לסטארטטים שוחצים 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 רגיסטרים לא עוזבים את הארגון.

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

3. **התאמה אישית** -- אפשר לעשות "Fine-tuning" ייודי.

2.2.2 שכבת החיבור: -- הרכזת של מודלים

הוא כמו שדה תעופה שמחבר אתכם לכל היעדים. במקומות לפתיחת חשבון בנפרד אצל OpenRouter, Anthropic, OpenAI, Google ו-100 מודלים שונים דרך API אחד. יתרונות מנהליים:

□ גמישות -- החלפת מודל זה שורת קוד אחת

□ השוואת עלויות -- תוכלו לנסוט 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 בשביבן. עם OpenRouter הם יישמו לוגיקה: פניות פשוטות ל-GPT-3.5 (אול), פניות מורכבות ל-Claude Sonnet (מדוק). חיסכיהם 60% בעליות בלי לוותר על איזמות.

3.2.2 שכבת הפיתוח: ספריות אינטגרציה

כדי לבנות מערכת אמיתית, לא מספיק לשלווח בקשה ל-API ול לקבל תשובה. צריך לנתח שיחות, לזכור הקשר, לחבר בסיסי נתונים, לטפל בשגיאות. כאן ננכשות הספריות:

-- הסוכן המקציעי LangChain

- LangChain היא הספרייה הפופולרית ביותר לבניית אפליקציות LLM. היא מציעה:
- -- **Chains** שרשור פעולה. לדוגמה: קח שאלה → חפש בסיס נתונים → שלח ל-LLM → עצב תשובה.
 - -- **Agents** סוכנים אוטונומיים שיודעים לבחור כלים. "איזה טיסות זולות לברלין?" → הסוכן מבין שצריך לקרוא ל-API של טיסות, מנתה תוכאות, ומחייב תשובה.
 - -- **Memory** זכר מה דיברתם לפני 10 הודעות.
 - -- **Retrievers** חיבור לבסיס נתונים וקטוריים (נדבר עליהם בהמשך).
- מתי להשתמש ב-LangChain?** כשאתם בונים משהו מורכב -- סוכן שירות, מערכת RAG, אווטומציה רב-שלבית.

-- תזמור תהליכי LangGraph

LangGraph הוא המשך של LangChain, מוקד בניהול תהליכי מורכבים עם מעברים ותנאים. דמיינו תהליך אישור הזמנה:

1. בדיקת מלאי
2. אישור מנהל (אם מעל 10,000 ש"ח)
3. שליחה לוגיסטיקה
4. עדכון לקוח

LangGraph מאפשר לכם לתוכנן את התהליך כgraf זרימה, עם צמתים (פעولات) וקשתות (תנאים). כל צומת יכול להיות LLM, שאלחת מסד נתונים, או קריאה חיצונית.

-- המובנה והמסודר Pydantic AI

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

- הגדרת מבני נתונים נוקשים
- וlidציה אוטומטית של תשובות LLM
- אכיפת פורמטים (JSON Schema)

דוגמא: אתם רוצים ש-LLM יחלץ מפניהית ללקוח: שם,מייל, נושא, רמת דחיפות. Pydantic AI מגדר מבנה קפדי, ואם ה-LLM מחזיר משהו שלא תואם -- יש שגיאה מיידית.

4.2.2 שכבת ההטמעה: Embeddings וمسדי נתונים וקטוריים

אחת התובנות המרכזיות בעולם-h-AI המודרני היא שמלילים חן 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 זהה. משפטים דומים במשמעותם קרובים גיאומטרית; משפטים שונים רחוקים.
לדוגמא:

- ”כלב“ ו-”גור“ -- קרובים מאוד
- ”כלב“ ו-”מכונית“ -- רחוקים

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

שאלה: ”איך מגשים תביעת ביטוח?“ **מסמך במערכת:** ”הליך הגשת דרישת לפיצויי“
חיפוש רגיל לא ימצא את זה -- אין מילים משותפות. חיפוש וקטורי כן.

מודלי Embedding מוביילים

- **OpenAI Text-Embedding-3** -- הסטנדרט. קל לשימוש, איזוט מצוינת. גרסה small (זולה) וגרסת large (מדוייקת).
- **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 לקטוריים, איפה נשגן אותם? מסד נתונים רגיל (PostgreSQL ,MySQL ,MySQL) לא יודע לעבוד עם חיפוש וקטורי. צריך מסד נתונים וקטורי (Vector Database).
Pinecone -- הענן המנוהל:

- **יתרונות:** לא צריך להתකין כלום. שירות ענן מנוהל, סקיילבילי, מהיר.
- **חסרונות:** עלות חודשית, תלות בספק.
- **מתי להשתמש:** כשאתם רוצים עלות מהר, בלי להתעסק בתשתיות.

Chroma -- הפתרון המקומי:

- **יתרונות:** קוד פתוח, חינמי, רץ על הרשת שלכם.
- **חסרונות:** אתם צריכים לנוהל: גיבויים, ביצועים, סקייל.
- **מתי להשתמש:** כשאתם בשלב POC, או כשאתם רוצים שליטה מלאה.

Weaviate -- הכליה היברידית:

- **יתרונות:** תומך גם בחיפוש וקטורי וגם בחיפוש טקסט רגיל. אינטגרציות עשירות.
- **חסרונות:** מורכב יותר להקמה.

□ **מתי להשתמש:** כאשרם צרכיים גמישות מקסימלית.

-- המהיר: Qdrant

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

□ **חסרונות:** קהילה קטנה יותר.

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

5.2.2 שכבת האוטומציה: כלים אגנטיים

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

(שוב, אבל בהקשר אחר) LangGraph

כבר דיברנו עליו בספריה, אבל הוא גם כלי תיזמור. LangGraph מאפשר לבנים תהליכי רב-שלביים שבהם סוכנים שונים מתקשרים זה עם זה, מעבירים מידע, ומקבלים החלטות.

-- צוותי סוכנים AutoGen

מבית Microsoft (Microsoft) הוא פרויימורק לבניית מערכות רב-סוכן. דמיינו שאתם בונים מערכת ניהול פרויקטים:

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

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

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

כל סוכן הוא LLM עם הנחיות (System Prompt) "יעודיות. מנהל את התקשרות בינהם.

ח8ח -- אוטומציה חזותית

ח8ח הוא כלי No-Code / Low-Code לאוטומציה. במקום לכתוב קוד, אתם גוררים בלוקים ומחברים אותם.

דוגמה:

1. כל פניה ללקוח במיל →

2. LLM מנתח ומטליג →

3. אם דוחף: שלוח SMS למנהל →

4. אם רגיל: פותח כרטיס ב-Jira

הכל בלי לכתוב שורת קוד אחת.

-- השחקן הוותיק -- Zapier

קיימים הרבה לפני AI, אבל הוא הש恬במצוין. תומך באלפי אינטגרציות (kcalS, liamG, OpenAI...). לאחרונה הוסיף תמיכה ב-AI-Anthropic .ecrofselaS ?Zapier מתי n8n ומתי

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

3.2 מתי להשתמש במה: מטריצת החלטות

עכשו שאנו מכירים את כל השחקנים, איך בוחרים? הנה מטריצה מנהלית:

1.3.2 בחירת ספק LLM

טבלה 1.2: מטריצת בחירת מודל שפה

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

2.3.2 בחירת מס' נתונים וקטורי

3.3.2 החלטת ענן מול On-Premise

פערון המטריצה:

- **רבייע שמאל תחתון (כחול):** נתונים לא רגישים, נפח נמוך → Cloud (Anthropic, OpenAI)
- **רבייע ימני תחתון (ירוק):** נתונים רגישים, נפח נמוך → Hybrid (חלק בענן, חלק מקומי)
- **רבייע ימני עליון (אדום):** נתונים רגישים, נפח גבוה → On-Prem (Llama 3) על שירותי פנימיים
- **רבייע שמאל 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 צרי DevOps	Pinecone	"יצור, סטארטאפ
שליטה מלאה, גמישות	Weaviate ו Qdrant	ארגון גדול, On-Prem
אופטימיזציה קיצונית	Qdrant	צורך ב מהירות מקסימלית

נפח שימוש חדש



אייר 2.2: מטריצת החלטה: ענן מול אונ-פרמיז

4.2 נסחאות מנהליות: חשבון כלכלי

כמו前所ים, אנחנו צריכים להצדיק כל החלטה טכנולוגית בשפה כלכלית. הנה שתי נסחאות קריטיות:

1.4.2 נסחת TCO -- עלות בעלות כוללת

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

פירוק הנסחה:

-- עלות רישי שנתי (מנויים ל-IPA, רישיונות תוכנה) C_{license} □

-- עלות תשתיות (שרטים, אחסון, רשת, חשמל) C_{infra} □

-- עלות כוח אדם חדשית (מפתחים, DevOps, מנהלי מערכת) C_{HR} □

דוגמה 1: פתרון ענן טהור (OpenAI)

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

□ תשתיות: \$0 (ספק מנוהל)

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

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

דוגמה 2: פתרון On-Premise (Llama 3)

□ רישיון: \$0 (קוד פתוח)

□ תשתיות: שרת עם 8 GPUs (\$80,000 לשנה, פחות 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: בטוחה הקצר, הענן זול יותר. אבל אם נפח השימוש גדול -- On-Premises משתלם יותר.

2.4.2 נוסחת -- זמן תגובה -- Latency

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

פירוק הנוסחה:

□ -- זמן העברת הבקשה והתשובה בראשת (TTR) T_{network}

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

□ -- זמן ההසקה של המודל עצמו T_{model}

דוגמה: שאילתת RAG

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

(Pinecone + Embedding) $100ms = T_{\text{processing}}$

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

סה"כ (ען):

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

סה"כ (מקומי):

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

מסקנה: זמן רשות נראה קטן, אבל בנפח גבוה (אלפי בקשות ביום) -- הוא ממשמעותי.

טבלה 3.2: השוואת מחירי מודלים מרכזיים

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

* משוקלל: הנחה של 50% קלט, 50% פלט

** דרך OpenRouter או ספקים אחרים

*** עלות אפסית לשימוש, אבל יש עלות תשתייה

5.2 השוואת מחירים: מי הכי משתמש?

מחירי API משתנים כל הזמן, אבל הנה תמונה מצב (נכון לדצמבר 2024):
תובנות מהטבלה:

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

6.2 דוגמאות מעשיות מהשיטה

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.
צרcis:

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

- אינטגרציה עם בנקים (API)

ארכיטקטורה מומלצת:

- GPT-3.5 לשאלות פשוטות, Claude Sonnet עם OpenRouter :**LLM** לモרכבות
- OpenAI Text-Embedding-3-small :**Embeddings**
- Pinecone תוכנית חינמית (עד 100K וקטורים) :**Vector DB**
- LangChain (קהילה גדולה, הרבה דוגמאות) :**Framework**
- (Self-Hosted) n8n :**Automation** (חינמי)

עלויות:

- OpenRouter :\$800/\$חודש (בממוחע)
- Pinecone :\$0 (תוכנית חינמית)
- AWS EC2 על Self-Hosted) \$0 :n8n (\$20/חודש)
- פיתוח: מפתח אחד חצי שעה
- סה"כ: \$820/\$חודש -- מתוחת לתכניב, עם מרוחה לדילה.

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

תרchysh: 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 לתחilibים של הארגון

ארכיטקטורה מומלצת:

- Azure OpenAI :**LLM** (גרסה ארגונית -- נתונים לא משמשים לאימון)
- Weaviate על שירותי פנימיים :**RAG**
- Azure OpenAI Embeddings :**Embeddings**
- LangSmith עם LangChain :**Framework** (ניתוח ולוגים)

UI: פורטל פנימי מבוסס

ת戎נות:

- נתונים נשארים ב-Tenant הארגוני של Azure
- אפשר לחבר לכל מסדי הנתונים הפנימיים

- לוגים מלאים לביקורת
- התאמת אישית -- אפשר לכוון את המודל לתהליכיים ספציפיים
 - עליות (הערכה):**
 - פיתוח :Azure OpenAI :\$/15,000 /חודש (500 משתמשים)
 - Weaviate על Azure VM :\$/2,000 /חודש
 - פיתוח ותחזקה: 2 מפתחים (\$20,000 /חודש)
- סה"כ: \$37,000 /חודש** -- נשמע יקר? לא בהשוואה ל-500 רישיונות ChatGPT Plus (\$10,000 /חודש) ללא שליטה ולא התאמת 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 חודשים עבודה של מפתח (\$30,000)
- **עלות ריצה:** API + שירותי (\$1,500 /חודש)
- **תחזקה:** רביע משירה (\$3,000 /חודש)
- **סה"כ שנה ראשונה:** $\$30,000 + (\$4,500 \times 12) = \$84,000$
- ניתוח Buy (Intercom):**
 - **רישיון:** \$5,000 /חודש (500 שימוש ביום)
 - **אינטגרציה:** חודש עבודה (\$10,000)
 - **תחזקה:** כמעט אפסית
 - **סה"כ שנה ראשונה:** $\$10,000 + (\$5,000 \times 12) = \$70,000$
 - ההחלטה:** בשנה הראשונה, Buy זול יותר. אבל
 - אם הנפח גדל (1,000 שימוש ביום) -- Intercom עולה ל-\$10,000 /חודש
 - פתרון Build נשאר באותה עלות (או עולה מעט)
 - פתרון Build מאפשר התאמת מלאה לתהליכיים
- המלצתה:** התחלו 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 סמנים'ל טכנולוגיות בחברת ביטוח בגיןית (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 רגילים
- זמן תגובה: עד 3 שניות
- תקציב: \$10,000/חודש
- 50 שאלות ביום בממוצע

משימה:

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

2.7.2 תרגיל 2 (תיאורטי): חישוב TCO לשלווה תרחישים

השו TCO לשלווה אפשרויות:

תרחיש א': ענן טהור

- OpenAI GPT-4

- Pinecone管

- מפתח חצי משראה

תרחיש ב': היברידי

- Azure OpenAI (ארగוני)

- Azure VM על Weaviate

- מפתח + מנהל מערכת (שני חצאי משראה)

תרחיש ג': On-Premise מלא

- Llama 3.1 70B על שרת פנימי

- Chroma מקומי

- מפתח + מנהל מערכת + DevOps

חשיבותו TCO לשנה לכל תרחיש. איזה משתמש לנפח של:

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

3.7.2 **תרגיל 3 (תיאורטי): ניתוחTrade-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**

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

1. הסבירו מהם הסיכוןים של תלות בספק בודד
2. תכננו אסטרטגיית Multi-Vendor (יוטר 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 לטוווח ארוך. עליהם לכתוב RFP (tseuqeR of rof lasoporP).

משימה: כתבו RFP שככל:

1. רקע על הארגון וצריכיו
2. דרישות פונקציונליות (מה המערכת צריכה לעשות)
3. דרישות לא-פונקציונליות (ביצועים, אבטחה, זמינות)
4. קритריוני הערכה (איך תבחרו בין ההצעות)
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 עבורו:

- GPT-4 Turbo □
- GPT-3.5 Turbo □
- Claude 3.5 Sonnet □
- Gemini Pro □

בונוס: 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
26     pricing = PRICING[model_name]
27
28     # Calculate total tokens per month
29     total_input = input_tokens * requests_per_month
30     total_output = output_tokens * requests_per_month
31
32     # Calculate cost (price per million, divide by million)
33     input_cost = (total_input / 1_000_000) * pricing["input"]
34     output_cost = (total_output / 1_000_000) * pricing["output"]
35
36     return input_cost + output_cost
37
38 # Usage example
39 if __name__ == "__main__":
40     # Example parameters
41     input_tokens = 500
42     output_tokens = 200
43     requests = 10_000 # 10K requests per month
44
45     print("Monthly cost comparison:")
46     print(f"Parameters: {input_tokens} input tokens, "
47           f"{output_tokens} output tokens, {requests:,} requests\n")
48
49     for model in PRICING.keys():
50         cost = calculate_monthly_cost(model, input_tokens,
51                                       output_tokens, requests)
52         print(f"{model}: ${cost:8.2f}")
53
54     # 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:

בדיקות של מודלים Latency

```

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 שלם לאקוסיסטם

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

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

למדנו לחשב TCO ו Latency, להשוות בין שירותי, 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 אמיהיות.

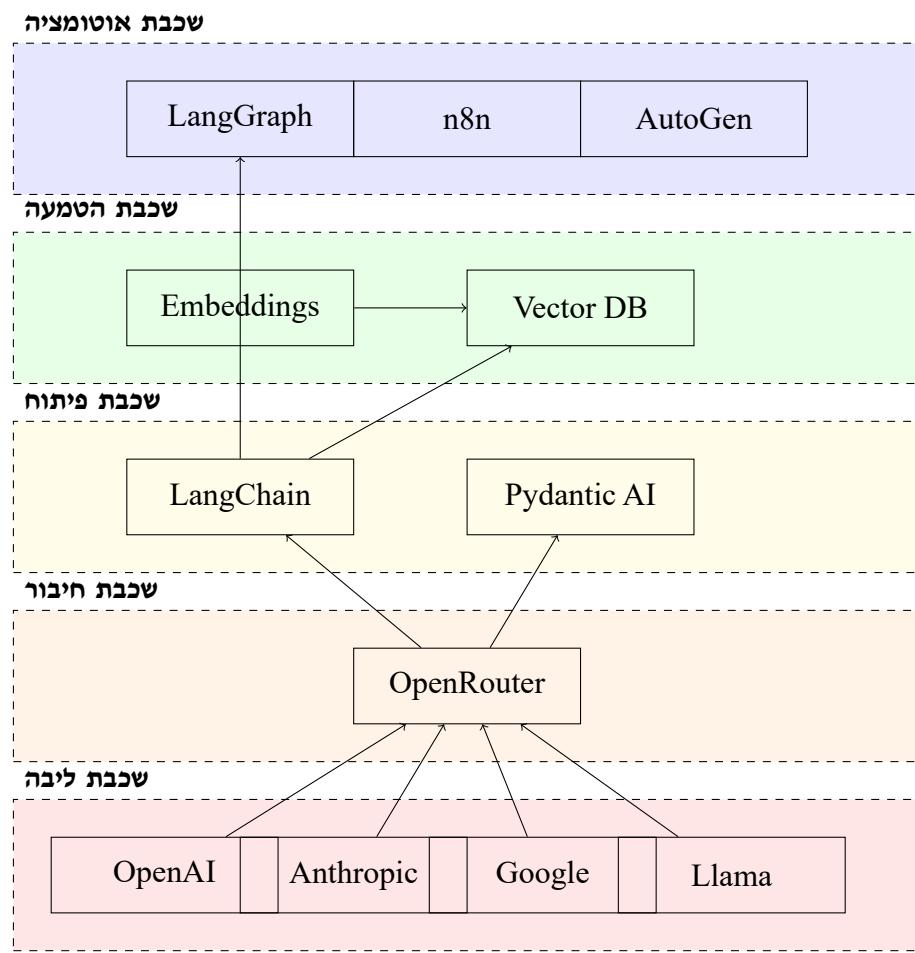
נספח: גרפים ותרשימים

תרשים 1: ארכיטקטורת אקוסיסטם מלאה

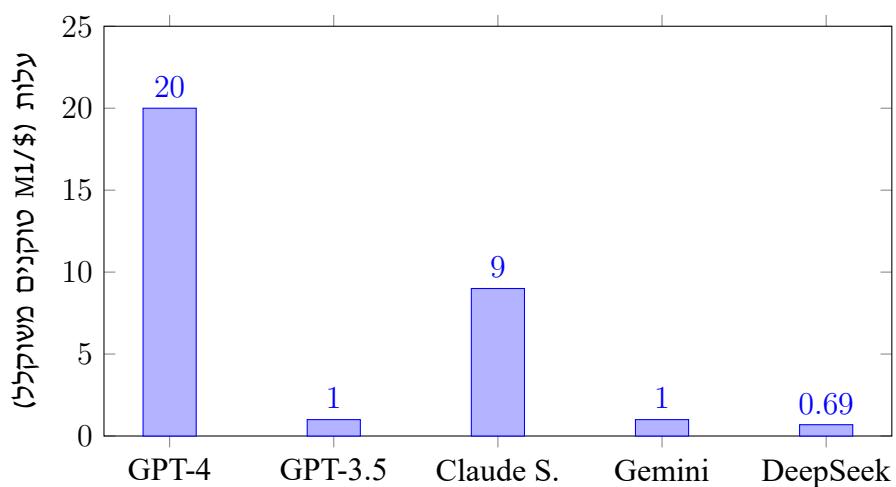
תרשים 2: גרפ' עמודות -- השוואת מחירים

רשימת קריאה מומלצת

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



איור 2.2: ארכיטקטורת אקויסיטטם AI מלאה -- 5 שכבות



איור 3.2: השוואת מחירים ממוצעים בין מודלים מוביילים