

# פרק 1

## שיקולים אסטרטגיים -- בחירת טכנולוגיות וניהול זיכרון

### מטרות הלמידה

בסיום פרק זה תהיו מסוגלים:

- לקבל החלטות מושכלות על בחירת LLM ו-Embedding Models המתאימים לצרכי הארגון
- להבין את אסטרטגיות ניהול זיכרון ב-LLM והשלכותיהן העסקיות
- לתכנן אסטרטגיית טכנולוגיה ארוכת טווח תוך הימנעות מ-Vendor Lock-in
- להעריך את היחס בין ביצועים לעלות במודלים שונים
- לנהל בצורה יעילה את מגבלות Context Window במערכות IA

### 1.1 פתיחה: עולם של בחירות

לפני מאתיים שנה, בעידן המהפכה התעשייתית, עמדו יזמים בפני שאלה פשוטה אך הרת גורל: האם לרכוש מנוע קיטור מסוג א' או מסוג ב'? ההחלטה הזו, שנראתה טכנית בלבד, קבעה לעתים קרובות את עתידו של בית המלאכה. מנוע יעיל מדי יכול היה ליצור עלויות תחזוקה מופרזות, בעוד מנוע חלש מדי לא יכול היה לענות על דרישות הייצור הגדלות. והכי גרוע מכל -- בחירה במנוע "לא נכון" מבחינה טכנולוגית יכולה להשאיר את בית המלאכה מאחור כאשר הסטנדרט התעשייתי השתנה.

כיום, מנהלים בעולם הבינה המלאכותית עומדים בפני דילמות מפתיעות באופן מדהים. לא מדובר במנועי קיטור, אלא במודלי שפה גדולים -- LLM. וכמו אותם יזמים מהמאה ה-19, גם מנהלי העידן הדיגיטלי חייבים לבחור בחוכמה, כי ההשלכות של בחירה מוטעית יכולות להיות כבדות: עלויות מופרזות, ביצועים נמוכים, תלות בלתי רצויה בספק יחיד, או גרוע מכל -- הצורך להתחיל מחדש כאשר הטכנולוגיה שבחרנו הופכת למיושנת.

בפרק זה נעסוק בשאלות האסטרטגיות המורכבות ביותר של הטמעת IA בארגון: איזה מודל שפה לבחור? איך לנהל את זיכרון השיחה? מתי כדאי להשקיע במודל cificepS-ksaT ומתי להישאר עם esopruP-lareneG? וכיצד לבנות ארכיטקטורה שלא תאלץ אותנו "להתחתן" עם ספק אחד לשארית ימי הפרויקט?

## 2.1 בחירת MLL: המפה האסטרטגית

### 1.2.1 קריטריונים מרכזיים לבחירת מודל

כאשר ארגון עומד בפני החלטה על בחירת LLM, הוא למעשה מקבל החלטה אסטרטגית רב-שנתית. ההחלטה הזו דומה יותר לבחירת ספק PRE מאשר לרכישת תוכנה פשוטה. הסיבה פשוטה: כל בחירת מודל מגיעה עם השלכות עמוקות על הארכיטקטורה, על הנתונים, ועל הצוות שיעבוד מולה.

לפני שאתם מתחילים להשוות ציונים ב-Benchmarks, שאלו את עצמכם:

1. מהן המשימות הספציפיות שהמודל צריך לבצע? (סיכום, יצירת תוכן, קוד, שיחה?)
2. מהי רמת הרגישות של הנתונים? (האם נתונים רגישים יכולים לעזוב את הארגון?)
3. מהי התקציב החודשי המוקצה? (עלות לטוקן □ נפח שימוש חזוי)
4. מהי רמת ה-Latency המקסימלית המקובלת? (זמן תגובה)
5. האם נדרשת תמיכה בשפות מרובות? (מעבר לאנגלית)
6. מהו גודל Context Window הנדרש? (כמה מידע צריך לעבד בו זמנית)

בואו ננתח כל קריטריון בנפרד.

#### ביצועים לפי סוג משימה

לא כל מודל מצטיין בכל משימה [21], [71]. GPT-4, למשל, מצטיין במשימות שדורשות הבנה עמוקה והיגיון מורכב, אך הוא יקר יחסית ואיטי. Claude 3.5 Sonnet, לעומת זאת, מציג יכולות מצוינות בכתיבה יצירתית וניתוח טקסט ארוך, תוך שהוא מהיר יותר ולעתים זול יותר. GPT-3.5 Turbo הוא זול במיוחד ומהיר, אך פחות מדויק במשימות מורכבות.

חברת ביטוח גדולה צריכה IA לשתי מטרות שונות:

1. תמיכת לקוחות בזמן אמת -- תגובות מהירות לשאלות נפוצות.
2. ניתוח תביעות מורכבות -- בדיקה עמוקה של מסמכים וזיהוי הונאות.

אסטרטגיה חכמה היא להשתמש ב-M-ituM ledom-ygetart:

□ עבור תמיכת לקוחות: GPT-3.5 Turbo (מהיר, זול, מספיק טוב)

□ עבור ניתוח תביעות: GPT-4 או Claude Opus (איטי אך מדויק)

התוצאה: חיסכון של 70% בעלויות IPA תוך שמירה על איכות גבוהה במשימות קריטיות.

#### רגישות נתונים ופרטיות

אם הארגון מעבד מידע רגיש -- רפואי, פיננסי, אישי -- השאלה "לאן הולכים הנתונים שלי?" הופכת קריטית. מודלים בענן כמו GPT-4 או Claude שולחים את הנתונים לשרתי הספק. אמנם IAnePO

ו-`ciPorhtnA` מבטיחים שנתונים לא משמשים לאימון נוסף, אך עבור ארגונים מסוימים אפילו זה לא מספיק טוב.

בתרחישים אלו, פתרונות **Self-Hosted** כמו **Llama 3.1 405B** או **Mistral Large** מאפשרים הרצה מלאה `sesimerP-nO`, כך שהנתונים לעולם לא עוזבים את הארגון. אמנם יש עלות אינפרסטרוקטורה (שרתים, UPG), אך לעתים זה המחיר ההכרחי של פרטיות.

$$\text{Total Privacy Cost} = \text{Infrastructure Cost} + \text{Maintenance} + \text{HR Cost}$$

לעומת:

$$\text{Cloud API Cost} = \text{Tokens} \times \text{Price per Token}$$

**נקודת האיזון** היא הנפח החודשי שבו שני המסלולים שווים בעלות.

#### גודל `wodniW txetnoC`

**Context Window** הוא מספר הטוקנים המקסימלי שהמודל יכול לעבד בשיחה אחת. זה כולל את כל ההיסטוריה של השיחה, את הפרומפט, ואת התשובה הצפויה.

□ `obruT 5.3-TPG`: K61 טוקנים (כ-21,000 מילים)

□ `obruT 4-TPG`: K821 טוקנים (כ-69,000 מילים)

□ `tennoS 5.3 edualC`: K002 טוקנים (כ-51,000 מילים)

□ `orP 5.1 inimeG`: M2 טוקנים (כ-5.1 מיליון מילים)

למה זה חשוב? אם אתם צריכים לעבד מסמכים ארוכים (חוזים, דוחות שנתיים, תיעוד טכני), `wodniW txetnoC` גדול חוסך את הצורך ב-**RAG** ובעיבוד רב-שלבי.

משרד עורכי דין צריך לנתח חוזים מורכבים באורך 5,000 מילים. אם הם משתמשים ב-`obruT 5.3-TPG`, הם חייבים לפצל את החוזה לחלקים קטנים ולשלוח כל חלק בנפרד -- זה יוצר פיצול הקשר וסיכון להחמצת קשרים בין סעיפים. פתרון: שימוש ב-`tennoS 5.3 edualC` עם `wodniW txetnoC K002` מאפשר לשלוח את כל החוזה בבת אחת, וכך המודל רואה את התמונה המלאה. **עלות לפי מודל (לחוזה בודד של K05 מילים = K76 טוקנים):**

□ `obruT 5.3-TPG`: פיצול ל-5 קריאות □  $K1/200.0\$ = 76.0\$$  (אך איכות נמוכה יותר)

□ `tennoS 5.3 edualC`: קריאה אחת □  $K1/300.0\$ = 02.0\$$  (איכות גבוהה יותר)

### 2.2.1 מדדי השוואה: `tsoC sv ecnamrofreP`

אחת הדרכים הטובות ביותר להשוות מודלים היא באמצעות `oitaR tsoC-ot-ecnamrofreP` [6], [9]. טבלה 1.1 מציגה השוואה בין המודלים המובילים, ואיור 1.1 מדגים את היחס בין ביצועים לעלות.

Performance/Cost Ratio =  $\frac{\text{Performance Score}}{\text{Monthly Cost}}$

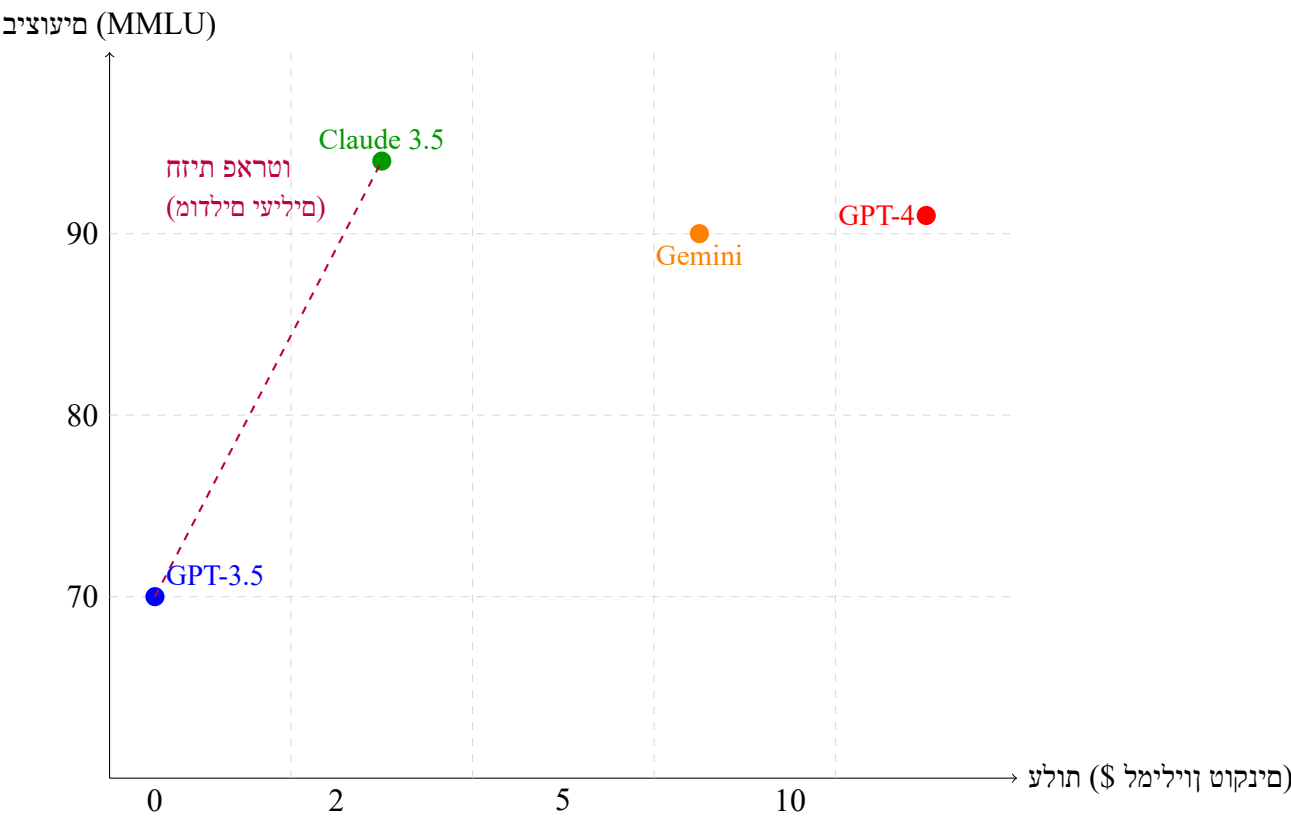
כאשר:

□ P erofreP : ציון מנורמל (0-100) ממדדים כמו LMM , lavEnamuH , או בדיקה פנימית

□ M ylhtnoM : עלות חודשית משוערת לפי נפח שימוש

טבלה 1.1: השוואת מודלים מובילים (5202)

לדומ	MMLU	קוט 1M/תולע	Context	Latency
GPT-4 Turbo	86.4	\$10	128K	3-5s
Claude 3.5 Sonnet	88.7	\$3	200K	2-4s
GPT-3.5 Turbo	70.0	\$0.50	16K	0.5-1s
Gemini 1.5 Pro	85.9	\$7	2M	4-6s
Llama 3.1 70B	79.3	Self-hosted	128K	1-2s



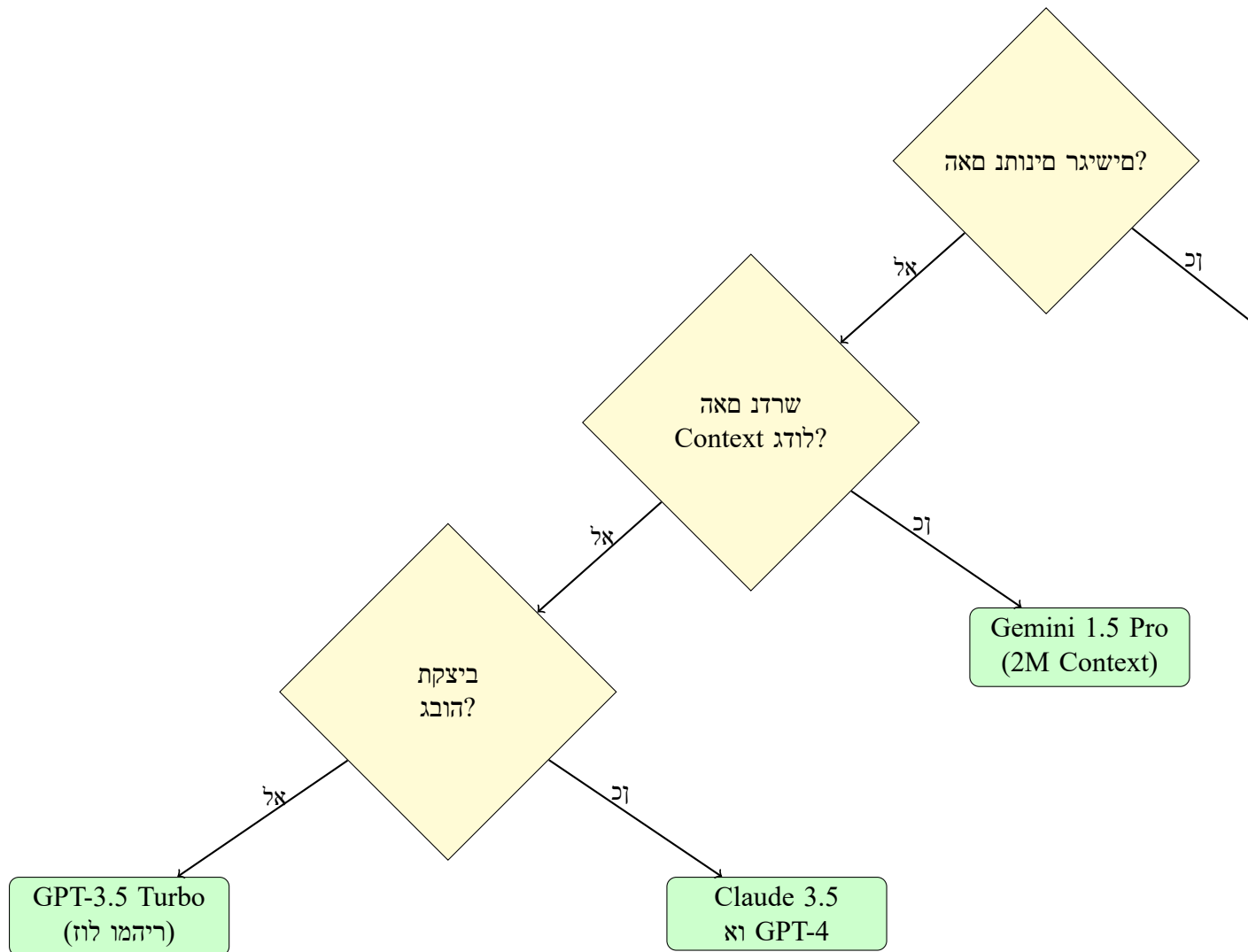
איור 1.1: גרף rettacS: ביצועים מול עלות

הגרף לעיל מדגים נקודה חשובה: tennoS 5.3 edualC נמצא על "חזית פארטו" -- הוא מציע

שילוב אופטימלי של ביצועים ועלות. מודלים שמעל הקו הסגול (כמו 4-TPG) יקרים יותר בלי לתת שיפור פרופורציונלי, ומודלים מתחת לקו (כמו 5.3-TPG) זולים אך פחות מדויקים.

### 3.2.1 eerT noisiceD לבחירת MLL

איור 2.1 מציג עץ החלטות פשוט לבחירת LLM על בסיס צרכי הארגון.



איור 2.1: eerT noisiceD לבחירת MLL

## 3.1 בחירת Embedding Models: OpenAI GPT-4 או OpenAI GPT-3.5?

Embedding Models הם המנוע שמאחורי מערכות RAG וחיפוש סמנטי [01], [31]. הם הופכים טקסט לווקטורים מספריים, מה שמאפשר למדוד "דמיון" בין משפטים, מסמכים או שאלות.

### 1.3.1 OpenAI GPT-4 vs OpenAI GPT-3.5

מודלים כמו text-embedding-3-large של OpenAI או NV-Embed-v2 של NVIDIA הם "מודלים כלליים" -- מאומנים על מגוון רחב של טקסטים ומסוגלים לטפל ברוב התחומים בצורה סבירה.

#### יתרונות:

- פשוט ליישום -- אין צורך באימון נוסף
- מתאים לרוב המקרים (80% מהמשימות)
- מצטיין בטקסטים כלליים ובשפה טבעית

#### חסרונות:

- פחות מדויק בתחומים מאוד ספציפיים (רפואה, משפטים, כימיה)
- עלויות IPA מתמשכות (אם משתמשים ב-IPA חיצוני)

### 2.3.1 sgniddebME cificepS-ksaT

במקרים שבהם הטקסט שלכם מאוד ספציפי -- למשל, מסמכים רפואיים, תקנות משפטיות, או קוד -- כדאי לשקול **Fine-Tuning** של מודל gndiddebME או שימוש במודל ייעודי.

חברת תרופות גדולה השתמשה ב-**text-embedding-3-large** לאחזור מאמרים מדעיים, אך גילתה שהדיוק נמוך -- המודל לא הבין טוב מונחים רפואיים כמו "yticixotorhpen" או "emorrhagic".

הפתרון: **enuT-eniF** של מודל **BGE-M3** על 000,05 מאמרים רפואיים.

#### תוצאה:

- דיוק אחזור עלה מ-56% ל-98%
  - עלות חד-פעמית: \$000,5 (אימון)
  - עלות שוטפת: \$002/חודש (detsoH-fleS)
  - לעומת: \$000,2/חודש (IPA של IAnePO)
- נקודת איזון: 5.2 חודשים.

## 4.1 בחירת esabataD: rotceV, lanoitaleR, או dirbyH?

כאשר בונים מערכת IA, אחת השאלות הקריטיות היא: איפה לאחסן את הנתונים?

### 1.4.1 sesabataD rotceV

**Vector Databases** כמו **Weaviate**, **Pinecone**, ו-**Qdrant** מותאמים לאחסון וחיפוש של sgniddebME [2], [11]. הם מאפשרים חיפוש לפי דמיון (**Similarity Search**) במהירות גבוהה.

#### מתי להשתמש:

- כאשר רוב השאלות הן סמנטיות ("מצא מסמכים דומים לזה")
- כאשר יש צורך ב-GAR
- כאשר הנתונים הם לא מובנים (טקסט חופשי, תמונות)

### 2.4.1 sesabataD lanoitaleR

מסדי נתונים יחסיים מסורתיים כמו PostgreSQL או MySQL טובים למידע מובנה: טבלאות, שורות, עמודות.  
**מתי להשתמש:**

- כאשר הנתונים מובנים (לקוחות, הזמנות, מלאי)
- כאשר יש צורך ב-ACID Transactions (עסקאות אטומיות)
- כאשר השאילתות הן מדויקות (NIOJ, EREHW)

### 3.4.1 hcaorppA dirbyH

גישה היברידית משלבת את שני העולמות: נתונים מובנים ב-LQS, sgniddebME ב-BD rotceV. טבלה 2.1 מסכמת את ההבדלים בין הגישות השונות.

אתר מסחר אלקטרוני רוצה להציע המלצות מותאמות אישית.  
**ארכיטקטורה היברידית:**

- LQSertsoP: פרטי לקוחות, הזמנות, מלאי (נתונים מובנים)
- enoceniP: sgniddebME של תיאורי מוצרים והיסטוריית גלישה (חיפוש סמנטי)

**תהליך המלצה:**

1. שלוף מ-LQSertsoP: היסטוריית רכישות של הלקוח
2. המר את זה ל-gniddebME וחפש ב-enoceniP: מוצרים דומים
3. שלוף מ-LQSertsoP: פרטי המוצרים שנמצאו (מחיר, מלאי)
4. הצג ללקוח

טבלה 2.1: השוואת סוגי sesabataD

וירטורק	Vector DB	Relational DB	Hybrid
יטנמס שופיח	יוצמ	דמתנ אל	יוצמ
SQL תותליאש	דמתנ אל	יוצמ	יוצמ
העמטה תובכרומ	תינוניב	הכומנ	ההובג
תולע	ההובג-תינוניב	הכומנ	ההובג
תוריהמ (Similarity)	דואמ הריהמ	תיטיא	הריהמ

## 5.1 ניהול זיכרון ב-MLL: mret-gnoL, mret-trohS, lanretxE

אחד האתגרים הגדולים ביותר בעבודה עם LLM הוא ניהול זיכרון השיחה [51], [81]. בניגוד לבני אדם, שזוכרים אינסוף שיחות קודמות, MLL "שוכח" הכל ברגע שהשיחה מסתיימת. בנוסף, גם בתוך שיחה אחת, יש מגבלה על כמות המידע שהוא יכול "לזכור" -- זה נקרא Context Window.

### 1.5.1 yromeM mret-trohS: ניהול השיחה הנוכחית

yromeM mret-trohS היא ההיסטוריה המיידית של השיחה הנוכחית. בכל פעם שאתם שולחים הודעה ל-MLL, אתם בעצם שולחים:

1. את כל ההודעות הקודמות בשיחה

2. את ההודעה החדשה

זה אומר שככל שהשיחה ארוכה יותר, כך אתם משלמים יותר -- כי אתם מעלים שוב ושוב את כל ההיסטוריה.

אם כל הודעה מוסיפה ממוצע של 001 טוקנים, ובשיחה יש 02 הודעות:

$$\text{סינקוט} = 100 \times \frac{20 \times (20 + 1)}{2} = 21,000$$

זה בגלל שההודעה ה-1 נשלחת 02 פעמים, ההודעה ה-2 נשלחת 91 פעמים, וכו'.

#### אסטרטגיות לחיסכון:

□ noitacnurT: חתוך הודעות ישנות כשהן חורגות מ-wodniW txetnoC

□ noitazirammuS: סכם כל 01 הודעות לפסקה אחת

□ wodniW gnidils: שמור רק את ה-N הודעות האחרונות

### 2.5.1 yromeM mret-gnoL: זיכרון בין שיחות

yromeM mret-gnoL הוא היכולת של המערכת "לזכור" משהו גם אחרי שהשיחה הסתיימה. למשל, אם לקוח אמר לך בשבוע שעבר "אני צמחוני", אתה רוצה שהסוכן ידע את זה גם בשיחה הבאה.

**דרכים ליישום yromeM mret-gnoL:**

1. esabataD של עובדות: שמור עובדות על המשתמש (שם, העדפות, היסטוריה) ב-LQS

2. esabataD rotceV לשיחות קודמות: שמור sgniddebmE של כל שיחה, וכשמתחילה שיחה חדשה -- אחזר שיחות רלוונטיות

3. yrotsiH dezirammuS: סכם את כל השיחות הקודמות ל-"תקציר משתמש" (2-3 פסקאות)

סטארטאפ בונה סוכן תמיכה ללקוחות. הם רוצים שהסוכן "יזכור" שיחות קודמות. **ארכיטקטורה:**

□ LQSertsoP: טבלה customers -- שם, מייל, העדפות

□ BDamorphC: sgniddebmE של כל שיחה עם customer\_id

□ noitazirammuS: בסוף כל שיחה, סכם אותה ושמור ב-LQS

#### תהליך:

1. לקוח מתחבר עם customer\_id=123



2. שאילתה ל-LQSergetsoP: שלוף העדפות בסיסיות

3. שאילתה ל-BDamorhC: מצא 3 שיחות רלוונטיות מהעבר (דמיון סמנטי לנושא הנוכחי)

4. בנה tpmorP: "משתמש 321 הוא צמחוני, בעבר התלונן על איחור במשלוח. הנה שיחות קודמות: "...

5. שלח ל-MLL

תוצאה: הלקוח מרגיש "מובן" ולא צריך לחזור על עצמו.

### 3.5.1 yromeM lanretxE: גישה לידע חיצוני

yromeM lanretxE הוא היכולת של ה-MLL לגשת למידע שלא נמצא בתוך ה-wodniW txetnoC [1], [3]. זה נעשה בדרך כלל דרך RAG או דרך Function Calling.

□ GAR: אחזר מסמכים רלוונטיים מ-BD rotceV והזרק אותם לפרומפט

□ gnillaC noitcnuF: אפשר ל-MLL לקרוא לפונקציות חיצוניות (IPA של MRC, PRE, מזג אוויר)

## 6.1 wodniW txetnoC: מגבלות ואסטרטגיות התמודדות

Context Window הוא המגבלה הקשיחה ביותר של MLL [8], [91]. אם השיחה שלך חורגת ממנו, המודל פשוט לא יכול לקבל את הקלט.

### 1.6.1 אסטרטגיות להתמודדות עם wodniW txetnoC מוגבל

gniknuhC והזרקה חוזרת

אם יש לך מסמך ארוך מדי (למשל, ספר של 002 עמודים), אתה יכול לפצל אותו לחלקים, לשלוח כל חלק בנפרד, ולסכם את התוצאות.  
תהליך:

1. חלק את המסמך ל-01 חלקים

2. שלח כל חלק: "סכם את החלק הזה"

3. אסוף את 01 הסיכומים

4. שלח סיכום סופי: "סכם את 01 הסיכומים האלה לסיכום אחד"

wodniW gnidilS עם סיכום

כאשר השיחה ארוכה מדי, אל תשמור את הכל -- שמור רק את ה-01 הודעות האחרונות, ו"סכם" את השאר לפסקה אחת.

דוגמה:

□ הודעות 05-1: סוכמו ל-"המשתמש שאל על מוצרים, הוא מעוניין בטלפונים"

□ הודעות 06-15: שמורות במלואן

כך אתה "זוכר" את העבר, אבל לא משלם עבור כל הטוקנים.

**שימוש במודל עם txetnoC wodniW גדול**

הדרך הפשוטה ביותר: עבור למודל עם txetnoC גדול יותר.

□ אם אתה משתמש ב-TPG-5.3 (K61), עבור ל-TPG-4-obruT (K821)

□ אם אתה צריך יותר, עבור ל-edualC-5.3 tennoS (K002) או inimeG 5.1 orP (M2)

זה יקר יותר, אבל לפעמים זה הכרחי.

**7.1 ni-kcoL rodneV: הסיכון הנסתר**

**Vendor Lock-in** הוא המצב שבו הארגון הופך תלוי לחלוטין בספק אחד, ומעבר לספק אחר כרוך בעלויות אדירות או בלתי אפשרי לחלוטין [7], [41].

**1.7.1 איך נוצר ni-kcoL rodneV במערכות IA?**

1. `sIPA yrateirporP`: שימוש ב-IPA ייעודי של ספק מסוים (למשל, `gpt-4-vision`) שאין לו חלופה בספקים אחרים

2. `sledoM denuT-eniF`: אימון מודל ייעודי ב-`IANepO` -- לא ניתן להעביר אותו ל-`ciporhtnA`

3. `kcoL tamroF ataD`: שמירת נתונים בפורמט ייעודי שקשה להעביר (למשל, `sgniddebme` של `IANepO` לא תואמים ל-`sgniddebme` של `erehoC`)

4. `ycnednepeD wolfkroW`: שימוש בכלים ייעוד של ספק (למשל, `niahCgnaL` עם `IANepO` בלבד)

**2.7.1 אסטרטגיות למניעת ni-kcoL rodneV**

**שכבת הפשטה** (`reyaL noitcartsbA`)

במקום לקרוא ישירות ל-`openai.ChatCompletion.create()`, בנה ממשק כללי שיכול לקרוא לכל ספק [4], [61].

Listing 1.1: Python: הטשפה תבכש

```

1 class LLMProvider:
2     def __init__(self, provider: str):
3         self.provider = provider
4
5     def generate(self, prompt: str) -> str:
6         if self.provider == "openai":
7             return self._openai_generate(prompt)
8         elif self.provider == "anthropic":
9             return self._anthropic_generate(prompt)
10        elif self.provider == "local":
11            return self._local_generate(prompt)
12        else:
13            raise ValueError(f"Unknown provider: {self.provider}")
14
15    def _openai_generate(self, prompt):
16        import openai

```

```

17     response = openai.ChatCompletion.create(
18         model="gpt-4",
19         messages=[{"role": "user", "content": prompt}]
20     )
21     return response.choices[0].message.content
22
23 def _anthropic_generate(self, prompt):
24     import anthropic
25     client = anthropic.Anthropic()
26     message = client.messages.create(
27         model="claude-3-5-sonnet-20241022",
28         max_tokens=1024,
29         messages=[{"role": "user", "content": prompt}]
30     )
31     return message.content[0].text
32
33 def _local_generate(self, prompt):
34     # Ollama or local model
35     import ollama
36     response = ollama.chat(model='llama3.1', messages=[
37         {'role': 'user', 'content': prompt}
38     ])
39     return response['message']['content']
40
41 # Usage
42 llm = LLMProvider(provider="openai") # Easy to switch!
43 result = llm.generate("מהו AI?")

```

כעת, אם תרצה לעבור מ-OpenAI ל-ciporhtnA, פשוט תשנה את הפרמטר provider -- ללא צורך בשינוי קוד נוסף.

#### ytetartS ledoM-itluM

במקום להתחייב למודל אחד, השתמש ב-ytetartS ledoM-itluM:

□ משימות קלות: TPG-5.3 obruT

□ משימות מורכבות: C edual-5.3 tennoS

□ משימות רגישות: L 1.3 amal (S-fleH-detso)

כך אתה לא תלוי בספק אחד, וגם מפזר סיכונים.

#### noissergeR ומבחני תיעוד

כל פעם שאתה משנה ספק, אתה רוצה לוודא שהמערכת עדיין עובדת. לכן, בנה noissergeR stseT:

Listing 1.2: Python: Regression תחבם

```

1 def test_llm_output():
2     test_cases = [
3         {"prompt": "What is 2+2?", "expected_substring": "4"},
4         {"prompt": "Translate 'hello' to Hebrew", "
expected_substring": "שלום"},

```

```

5 ]
6
7 providers = ["openai", "anthropic", "local"]
8
9 for provider in providers:
10     llm = LLMProvider(provider=provider)
11     for case in test_cases:
12         result = llm.generate(case["prompt"])
13         assert case["expected_substring"] in result, \
14             f"Failed for {provider}: {result}"
15     print(f"{provider} passed all tests!")
16
17 test_llm_output()

```

### 3.7.1 חישוב tsoC gnihctiwS

לפני שמחליטים לעבור ספק, חשוב לחשב את עלות המעבר:

$$\text{Switching Cost} = C_{\text{dev}} + C_{\text{data}} + C_{\text{downtime}} + C_{\text{training}}$$

כאשר:

□  $C_{\text{dev}}$ : עלות פיתוח מחדש (שעות מהנדס □ שכר שעתי)

□  $C_{\text{data}}$ : עלות העברת נתונים והמרת sgniddebme

□  $C_{\text{downtime}}$ : אובדן הכנסות במהלך המעבר

□  $C_{\text{training}}$ : הדרכת הצוות על הכלי החדש

חברה משתמשת ב-TPG IAnepO-4 ורוצה לעבור ל-C edual 5.3 tennoS.  
חישוב:

□ פיתוח מחדש: 3 מהנדסים □ 2 שבועות □ 08 שעות □  $001\$ / \text{שעה} = 000,84\$$

□ העברת נתונים: BG005 sgniddebme □  $BG/20.0\$ = 000,01\$$  (צריך ליצור מחדש עם sgniddebme אחר)

□  $000,51\$$ : 3 ימים □  $000,5\$ / \text{יום הכנסות} = 000,51\$$

□ הדרכה: 02 עובדים □ 4 שעות □  $08\$ / \text{שעה} = 004,6\$$

$$\text{Switching Cost} = 48,000 + 10,000 + 15,000 + 6,400 = \$79,400$$

החלטה: אם המעבר ל-C edual חוסך  $000,3\$ / \text{חודש}$ , נקודת האיזון היא 5.62 חודשים. האם זה כדאי? תלוי באסטרטגיה ארוכת הטווח.

## 8.1 דוגמאות מעשיות

### 1.8.1 דוגמה 1: מעבר מ-TPG-5.3 ל-TPG-4

**תרחיש:** חברת SaaS השתמשה ב-TPG-5.3 obruT לסיכום פניות לקוחות. הם קיבלו תלונות על דיוק נמוך -- המודל "החמיץ" נקודות חשובות.  
**שיקולים:**

□ **ביצועים:** 4-TPG מדויק יותר, אך איטי פי 2-3

□ **עלות:** 4-TPG יקר פי 02 (!)

□ **נפח:** 000,01 פניות/חודש, ממוצע 005 טוקנים לפנייה

**חישוב עלויות:**

□ obruT 5.3-TPG:  $10 \times 0.5K \times \$0.002 = \$10$  /חודש

□ 4-TPG:  $150 \times 0.5K \times \$0.03 = \$150$  /חודש

**החלטה:** החברה עברה ל-dirbyH ygetartS:

□ 08% פניות "רגילות": obruT 5.3-TPG (\$8/חודש)

□ 02% פניות "מורכבות": 4-TPG (\$03/חודש)

עלות כוללת: \$83/חודש -- שיפור איכות משמעותי בעלות סבירה.

### 2.8.1 דוגמה 2: בחירה בין edualC ל-TPG לתמיכת לקוחות

**תרחיש:** סטארטאפ בונה סוכן תמיכה לקוחות אוטומטי. הם צריכים להחליט: edualC 5.3 tennoS או 4-TPG?

**בדיקת COP:** הם הריצו 001 שיחות אמיתיות במקביל על שני המודלים.  
**תוצאות:**

דדמ	Claude 3.5	GPT-4
תובנות קויד	87%	84%
עצומת הבוגת ןמז	2.3s	4.1s
החיש/תולע	\$0.015	\$0.08
תוחוקל ןוצר תועיבש	4.2/5	4.0/5

**החלטה:** edualC 5.3 tennoS -- מהיר יותר, זול יותר, ודיוק טוב יותר. 4-TPG לא הצדיק את העלות.

### 3.8.1 דוגמה 3: תכנון אסטרטגיית ledoM-itluM

**תרחיש:** ארגון גדול רוצה לבנות מערכת IA שמטפלת במגוון משימות.  
**ארכיטקטורה:**

□ QAF ושאלות פשוטות: obruT 5.3-TPG (מהיר וזול)

□ ניתוח חוזים ומסמכים: edualC 5.3 tennoS (txetnoC גדול, דיוק גבוה)

□ **נתונים רגילים:** detsoH-fleS B07 1.3 amall (פרטיות מלאה)

□ **יצירת קוד:** 4-TPG (מצטיין בקוד)

**תוצאה:**

□ עלות חודשית: \$005,2 (לעומת \$000,8 אם היו משתמשים רק ב-4-TPG)

□ איכות: גבוהה בכל תחום

□ גמישות: אפשר להחליף ספק בכל תחום בנפרד

## 9.1 תרגילים

### 1.9.1 תרגיל תיאורטי 1: בניית dracerocS להשוואת sMLL

**משימה:** בנה dracerocS להשוואת 4 מודלים: 4-TPG, 5.3 edualC, 5.1 inimeG, orP 5.1, B07 1.3 amall. **קריטריונים:**

□ ביצועים (ULMM)

□ עלות למיליון טוקן

□ wodniW txetnoC

□ ycnetaL

□ רגישות נתונים (האם detsoH-fleS?)

תן ציון 1-10 לכל קריטריון, ושקלל לפי חשיבות לארגון שלך.  
**פתרון לדוגמה:**

(לקשמ) ווירטירק	GPT-4	Claude 3.5	Gemini	Llama 3.1
סיעוציב (30%)	9	10	8	7
תולע (25%)	3	7	5	10
Context (20%)	7	9	10	7
Latency (15%)	5	7	4	8
תויטרפ (10%)	2	2	2	10
<b>ללוב וויצ</b>	<b>6.3</b>	<b>8.0</b>	<b>6.7</b>	<b>8.0</b>

**המלצה:** 5.3 edualC או 1.3 amall -- תלוי אם אתה מוכן לנהל detsoH-fleS.

### 2.9.1 תרגיל תיאורטי 2: חישוב tsoC gnihctiwS

**תרחיש:** חברה משתמשת ב-enoceniP (BD rotceV) ורוצה לעבור ל-tnardQ (detsoH-fleS). **נתונים:**

□ 01 מיליון sgniddebM ב-enoceniP

□ עלות enoceniP: \$005/חודש

□ עלות  $\text{detsoH-fleS tnardQ}$ : \$002/חודש (שרת)

□ זמן העברה משוער: 3 שבועות (2 מהנדסים)

□ שכר מהנדס: \$021/שעה

**חשב:**

1.  $\text{tsoC gnihctiwS}$

2. נקודת איזון (כמה חודשים עד שהמעבר משתלם?)

3. האם כדאי?

**פתרון:**

$$C_{\text{dev}} = 2 \times 3 \times 40 \times 120 = \$28,800$$

$$C_{\text{data}} = 10M \times \$0.0001 = \$1,000$$

$$C_{\text{downtime}} = 2 \times \$2,000 = \$4,000$$

$$C_{\text{training}} = 5 \times 8 \times 100 = \$4,000$$

$$\text{Total} = \$37,800$$

חיסכון חודשי:  $500 - 200 = \$300$

נקודת איזון:  $37,800/300 = 126$  חודשים (5.01 שנים!)

**החלטה:** לא כדאי --  $\text{tsoC gnihctiwS}$  גבוה מדי.

### 3.9.1 תרגיל תיאורטי 3: תכנון אסטרטגיית $\text{tnemeganaM txetnoC}$

**תרחיש:** אתה בונה סוכן שיחה ללקוחות. שיחה ממוצעת היא 03 הודעות, כל הודעה 051 טוקנים.

**בעיה:** אם תשלח את כל ההיסטוריה בכל פעם, תשלם הרבה!

**משימה:** תכנן 3 אסטרטגיות  $\text{tnemeganaM txetnoC}$  וחשב את העלות של כל אחת.

**פתרון:**

1.  $\text{yrotsiH lluf}$ : שלח הכל בכל פעם

$$\square \text{ טוקנים לשיחה: } 150 \times \frac{30 \times 31}{2} = 69,750$$

$$\square \text{ עלות: } 69,750 \times \$0.002/1000 = \$0.14$$

2.  $\text{wodniW gnidilS}$  (01 הודעות אחרונות)

$$\square \text{ טוקנים לשיחה: בממוצע } 150 \times 10 \times 30 = 45,000$$

$$\square \text{ עלות: } 45,000 \times \$0.002/1000 = \$0.09$$

3.  $\text{noitazirammuS}$  (סכם כל 01 הודעות)

$$\square \text{ טוקנים לשיחה: } 500 (\text{summary}) + 1,500 (\text{last } 10) = 2,000$$

$$\square \text{ עלות: } 2,000 \times \$0.002/1000 = \$0.004$$

**המלצה:**  $\text{noitazirammuS}$  -- חיסכון של 79%!

### 4.9.1 תרגיל תיאורטי 4: ניתוח ni-kcoL rodneV

**תרחיש:** בדוק את הארכיטקטורה הבאה וזהה נקודות ni-kcoL rodneV:

- dnetnorF: שולח בקשות ישירות ל-IAnepO IPA
- sgniddebme: משתמש ב-text-embedding-ada-002
- esabataD: שמור ב-enoceniP (ענן)
- ledoM denuT-eniF: ft:gpt-3.5-turbo:company:v1

**משימה:**

1. זהה 4 נקודות ni-kcoL

2. הצע פתרון לכל נקודה

**פתרון:**

1. ni-kcoL #1: dnetnorF קורא ישירות ל-IAnepO
  - **פתרון:** הוסף reyaL noitcartsbA (yxorP dnekcab)
2. ni-kcoL #2: sgniddebme ייעודיים ל-IAnepO
  - **פתרון:** השתמש במודל sgniddebme נייטרלי (detsoH-fleS 3M-EGB)
3. ni-kcoL #3: enoceniP בענן
  - **פתרון:** עבור ל-tnardQ או etaivaeW (detsoH-fleS)
4. ni-kcoL #4: ledoM denuT-eniF ייעודי
  - **פתרון:** שמור את gniniarT ataD -- אפשר לאמן מחדש על מודל אחר

### 5.9.1 תרגיל תיאורטי 5: בניית pamdaoR טכנולוגי ל-3 שנים

**תרחיש:** אתה OTC של סטארטאפ בתחום ה-hceTniF. בנה pamdaoR טכנולוגי IA ל-3 שנים.

**שלב 1 (שנה 1):**

□ COP עם IPA 4-TPG (מהיר ליישום)

□ BD rotceV בענן (enoceniP)

□ 000,1 לקוחות

**שלב 2 (שנה 2):**

□ מעבר ל-ledoM-itluM (4-TPG + edualC)

□ הוספת detsoH-fleS sgniddebme ledoM (3M-EGB)

□ 000,01 לקוחות

**שלב 3 (שנה 3):**

□ detsoH-fleS MLL (4 amalL) לנתונים רגישים

□ detsoH-fleS BD rotceV (tnardQ)

□ 000,001 לקוחות

**תוצאה:** גמישות מלאה, עלויות מבוקרות, אין ni-kcoL rodneV.



## 6.9.1 תרגיל קוד 6: השוואת ביצועי מודלים אוטומטית

משימה: כתוב סקריפט nohtyP שמשווה אוטומטית את הביצועים של 3 מודלים על 01 שאלות.

Listing 1.3: Python: מילדום יעוציב תאוושה

```

1 import time
2 from typing import List, Dict
3 import openai
4 import anthropic
5 import ollama
6
7 class ModelBenchmark:
8     def __init__(self):
9         self.results = []
10
11     def benchmark_openai(self, prompt: str) -> Dict:
12         start = time.time()
13         response = openai.ChatCompletion.create(
14             model="gpt-3.5-turbo",
15             messages=[{"role": "user", "content": prompt}]
16         )
17         latency = time.time() - start
18         answer = response.choices[0].message.content
19         tokens = response.usage.total_tokens
20         cost = tokens * 0.002 / 1000 # $0.002 per 1K tokens
21
22         return {
23             "model": "GPT-3.5 Turbo",
24             "answer": answer,
25             "latency": latency,
26             "tokens": tokens,
27             "cost": cost
28         }
29
30     def benchmark_claude(self, prompt: str) -> Dict:
31         client = anthropic.Anthropic()
32         start = time.time()
33         message = client.messages.create(
34             model="claude-3-5-sonnet-20241022",
35             max_tokens=1024,
36             messages=[{"role": "user", "content": prompt}]
37         )
38         latency = time.time() - start
39         answer = message.content[0].text
40         tokens = message.usage.input_tokens + message.usage.
41         output_tokens
42         cost = (message.usage.input_tokens * 0.003 +
43                 message.usage.output_tokens * 0.015) / 1000
44
45         return {
46             "model": "Claude 3.5 Sonnet",
47             "answer": answer,

```

```

47         "latency": latency,
48         "tokens": tokens,
49         "cost": cost
50     }
51
52     def benchmark_ollama(self, prompt: str) -> Dict:
53         start = time.time()
54         response = ollama.chat(model='llama3.1', messages=[
55             {'role': 'user', 'content': prompt}
56         ])
57         latency = time.time() - start
58         answer = response['message']['content']
59
60         return {
61             "model": "Llama 3.1 (Local)",
62             "answer": answer,
63             "latency": latency,
64             "tokens": 0, # Local - no token tracking
65             "cost": 0 # Self-hosted - no API cost
66         }
67
68     def run_benchmark(self, prompts: List[str]):
69         for i, prompt in enumerate(prompts):
70             print(f"\n=== Question {i+1}: {prompt[:50]}... ===")
71
72             # Test all models
73             for benchmark_func in [self.benchmark_openai,
74                                   self.benchmark_claude,
75                                   self.benchmark_ollama]:
76                 try:
77                     result = benchmark_func(prompt)
78                     self.results.append({
79                         "question": i+1,
80                         **result
81                     })
82                     print(f"{result['model']}: "
83                           f"{result['latency']:.2f}s, "
84                           f"${result['cost']:.4f}")
85                 except Exception as e:
86                     print(f"Error with {benchmark_func.__name__}: {
87                         e}")
88
89             self.print_summary()
90
91     def print_summary(self):
92         print("\n=== SUMMARY ===")
93         models = set([r['model'] for r in self.results])
94
95         for model in models:
96             model_results = [r for r in self.results if r['model']
97                             == model]

```

```

96     avg_latency = sum([r['latency'] for r in model_results
97 ]) / len(model_results)
98     total_cost = sum([r['cost'] for r in model_results])
99
100     print(f"\n{model}:")
101     print(f"    Avg Latency: {avg_latency:.2f}s")
102     print(f"    Total Cost: ${total_cost:.4f}")
103
104 # Usage
105 benchmark = ModelBenchmark()
106
107 test_prompts = [
108     "What is 2+2?",
109     "Explain quantum computing in simple terms.",
110     "Write a short poem about AI.",
111     "Translate 'Hello World' to Hebrew.",
112     "What are the benefits of cloud computing?",
113     "Summarize the French Revolution in 2 sentences.",
114     "What is the capital of Australia?",
115     "Explain the difference between AI and ML.",
116     "Write a Python function to calculate factorial.",
117     "What are the main causes of climate change?"
118 ]
119 benchmark.run_benchmark(test_prompts)

```

#### תוצאה צפויה:

=== YRAMMUS ===

obruT 5.3-TPG  
s2.1 :ycnetaL gvA  
0510.0\$ :tsoC latoT

tennoS 5.3 edualC  
s8.2 :ycnetaL gvA  
0230.0\$ :tsoC latoT

(lacoL) 1.3 amall  
s8.0 :ycnetaL gvA  
0000.0\$ :tsoC latoT

### 7.9.1 תרגיל קוד 7: ניהול זיכרון שיחה עם סיכום

**משימה:** כתוב מערכת שמנהלת זיכרון שיחה ארוכה באמצעות סיכומים אוטומטיים.

Listing 1.4: Python: מוכים מע נורכיז לזהינ

```

1 import openai
2 from typing import List, Dict
3
4 class ConversationMemoryManager:

```

```

5     def __init__(self, max_messages: int = 10):
6         self.messages: List[Dict] = []
7         self.summary: str = ""
8         self.max_messages = max_messages
9
10    def add_message(self, role: str, content: str):
11        """Add a new message to conversation history"""
12        self.messages.append({"role": role, "content": content})
13
14        # If exceeded max messages, summarize and truncate
15        if len(self.messages) > self.max_messages:
16            self._summarize_and_truncate()
17
18    def _summarize_and_truncate(self):
19        """Summarize old messages and keep only recent ones"""
20        print("□ Summarizing old messages...")
21
22        # Take first half of messages to summarize
23        messages_to_summarize = self.messages[:self.max_messages //
24]
25
26        # Create summary prompt
27        conversation_text = "\n".join([
28            f"{msg['role']}: {msg['content']}"
29            for msg in messages_to_summarize
30        ])
31
32        summary_prompt = f"""Summarize this conversation in 2-3
33sentences:
34
35{conversation_text}
36
37Previous summary: {self.summary if self.summary else 'None'}
38
39Provide a concise summary that captures key points."""
40
41        # Generate summary
42        response = openai.ChatCompletion.create(
43            model="gpt-3.5-turbo",
44            messages=[{"role": "user", "content": summary_prompt}],
45            max_tokens=200
46        )
47
48        new_summary = response.choices[0].message.content
49
50        # Update summary
51        if self.summary:
52            self.summary = f"{self.summary}\n\n{new_summary}"
53        else:
54            self.summary = new_summary

```

```

54     # Keep only recent messages
55     self.messages = self.messages[self.max_messages // 2:]
56
57     print(f"□ Summary updated. Keeping {len(self.messages)}
recent messages.")
58
59     def get_context_for_llm(self) -> List[Dict]:
60         """Get full context to send to LLM"""
61         context = []
62
63         # Add summary as system message if exists
64         if self.summary:
65             context.append({
66                 "role": "system",
67                 "content": f"Previous conversation summary:\n{self.
summary}"
68             })
69
70         # Add recent messages
71         context.extend(self.messages)
72
73         return context
74
75     def chat(self, user_message: str) -> str:
76         """Send message and get response"""
77         # Add user message
78         self.add_message("user", user_message)
79
80         # Get context
81         context = self.get_context_for_llm()
82
83         # Call LLM
84         response = openai.ChatCompletion.create(
85             model="gpt-3.5-turbo",
86             messages=context
87         )
88
89         assistant_message = response.choices[0].message.content
90
91         # Add assistant message
92         self.add_message("assistant", assistant_message)
93
94         return assistant_message
95
96     def get_stats(self) -> Dict:
97         """Get memory statistics"""
98         total_tokens = sum([
99             len(msg['content'].split()) * 1.3 # rough estimate
100             for msg in self.messages
101         ])
102

```

```

103         return {
104             "total_messages": len(self.messages),
105             "summary_length": len(self.summary.split()) if self.
summary else 0,
106             "estimated_tokens": int(total_tokens)
107         }
108
109 # Usage Example
110 memory = ConversationMemoryManager(max_messages=10)
111
112 # Simulate long conversation
113 conversation = [
114     "What is machine learning?",
115     "Can you give me an example?",
116     "How does it differ from deep learning?",
117     "What are neural networks?",
118     "Explain backpropagation.",
119     "What is gradient descent?",
120     "How do you prevent overfitting?",
121     "What is cross-validation?",
122     "Explain the bias-variance tradeoff.",
123     "What are ensemble methods?",
124     "Tell me about random forests.",
125     "How does XGBoost work?",
126     "What is feature engineering?",
127     "Explain dimensionality reduction.",
128     "What is PCA?"
129 ]
130
131 for user_msg in conversation:
132     print(f"\n User: {user_msg}")
133     response = memory.chat(user_msg)
134     print(f" Assistant: {response[:100]}...")
135
136 # Print stats every 5 messages
137 if len(memory.messages) % 5 == 0:
138     stats = memory.get_stats()
139     print(f"\n Stats: {stats}")
140
141 # Final summary
142 print("\n" + "="*50)
143 print("FINAL SUMMARY:")
144 print(memory.summary)
145 print("\n Final Stats:", memory.get_stats())

```

### הסבר הקוד:

1. המערכת שומרת רק את ה-01 הודעות האחרונות
2. כשעוברים את המגבלה, היא מסכמת את המחצית הראשונה
3. הסיכום נשלח כ-egasseM metasy בכל קריאה

4. כך חוסכים טוקנים אבל שומרים על הקשר

## 01.1 סיכום

בפרק זה למדנו כיצד לקבל החלטות אסטרטגיות בעולם ה-IA העסקי [5]. ראינו שבחירת **LLM** אינה רק שאלה טכנית -- היא החלטה עסקית שמשפיעה על עלויות, ביצועים, פרטיות וגמישות עתידית.

### נקודות מפתח:

□ **אין מודל "מושלם"** -- כל מודל מצטיין במשימות מסוימות. `ygetartS ledoM-itluM` היא לעתים קרובות הפתרון הטוב ביותר.

□ `wodniW txetnoC` הוא **מגבלה קריטית** -- תכנן מראש איך אתה מנהל זיכרון, ובחר מודל עם `txetnoC` מתאים למשימות שלך.

□ `ni-kcoL rodneV` הוא **סיכון אמיתי** -- השקיעו בשכבות הפשטה ובארכיטקטורה גמישה מהיום הראשון.

□ **חשבו ארוך טווח** -- `tsoC gnihctiwS` יכול להיות עצום. תכננו `pamdaoR` טכנולוגי ל-3 שנים לפחות.

בפרק הבא נעסוק בממשקי משתמש ואינטראקציה -- איך להפוך את כל הטכנולוגיה הזו לחוויית משתמש מעולה.