

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
- לנחל בצורה ייעילה את מגבלות **Context Window** במערכות AI

1.1 פתיחה: עולם של בחירות

לפני מאיים שנה, בעידן המהפכה התעשייתית, עמדו יזמים בפני שאלה פשוטה אך הרת גורל: האם לרכוש מנוע קיטור מסווג'A' או מסווג'B'? ההחלטה הזאת, שנراתה טכנית בלבד, קבעה לעיתים קרובות את עתידו של בית המלאכה. מנועiesel מידי יכול היה ליצור עליות תחזקה מופרזות, בעוד מנוע חלש מידי 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 מפתחיות באופן מדהים. לא מדובר במנועי קיטור, אלא במודלי שפה גדולים -- **LLM**. וכמו אוטם יזמים מהמאה ה-91, גם מנהלי העידן הדיגיטלי חווים לבחור בחוכמה, כי ההצלחות של בחירה מוצעת יכולות להיות כבדות: עליות מופרזות, ביצועים נמוכים, תלות בלתי רציה בספק יחיד, או גרוע מכל -- הצורך להתחילה מחדש הטכנולוגיה שהחרנו הופכת למיאשנה.

פרק זה נסוק בשאלות האסטרטגיות המורכבות ביותר של הטמעת AI בארגון: איזה מודל שפה לבחור? איך לנחל את זיכרון השיחות? متى כדאי להשקיע במודל T-ksaS-cepfcice? ומהי להישאר עם G-lareneP-esopru? וכי怎 לבנות ארכיטקטורה שלא תאלץ אותנו להתחתקן עם ספק אחד לשאריתימי הפרויקט?

2.1 בחרית TLL: המפה האסטרטגית

1.2.1 קритריונים מרכזיים לבחירת מודל

כאשר ארגון עומד בפני החלטה על בחרית LLM, הוא למעשה מקבל החלטה אסטרטגית רב-שנתית. ההחלטה זו דומה יותר לבחירת ספק PRE מאשר לרכישת תוכנה פשוטה. הסיבה פשוטה: כל בחרית מודל מגיעה עם השЛОות عمוקות על הארכיטקטורה, על הנתונים, ועל הוצאות שיעבוד מולה.

לפניהם מתחלים להשוות ציונים ב-Benchmarks, שאלו את עצמכם:

1. מהן המשימות הספציפיות שהמודל צריך לבצע? (סיכום, יצירה תוכן, קוד, שיחה?)
2. מהי רמת הרגישות של הנתונים? (האם נתונים רגילים 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 נדרש תמייה בשפות מרובות? (מעבר לאנגלית)
6. מהו גודל Context Window הנדרש? (כמה מידע צריך לעבד בו זמן)

באו ננתח כל קритריון בנפרד.

ביצועים לפי סוג משימה

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

חברת ביטוחגדולה צריכה AI לשתי מטרות שונות:

1. **תמיכת ל��וחות בזמן אמיתי** -- תשובות מהירות לשאלות נפוצות.
2. **ניתוח תביעות מורכבות** -- בדיקה עמוקה של מסמכים ויזיה הונאות.

סטרטגייה חכמה היא להשתמש ב-GPT-3.5 Turbo.

□ עבור תמיכת ל��וחות: GPT-3.5 Turbo (מהיר, זול, מספק טוב)

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

התוצאה: חיסכון של 70% בעלות IPA תוך שמירה על איכות גבוהה במשימות קרייטיות.

региשות נתונים ופרטיות

אם הארגון מעבד מידע רפואי -- רפואי, פיננסי, אישי -- השאלה "לאן הולכים הנתונים שלי?" הופכת קריטית. מודלים בענן כמו GPT-4 או Claude Sholchim את הנתונים לשרתים הספק. אמן OAIAnep

פרק 1. שיקולים אסטרטגיים -- בחירת טכנולוגיות ומילוי זיכרון

1- מושגים מבטחים שנדרנים לא משמשים לאימון נוספת, אך עבור ארגונים מסוימים אפילו זה לא מספיק טוב.

בתרחישים אלו, פתרונות **Self-Hosted** כמו **Mistral Large Llama 3.1 405B** או **Mistral Large** עשויים הרצה מלאה O-nP-sesimer, כך שהנתונים לעולם לא עוזבים את הארגון. אמנם יש עלות אינפרסטראטורה (שירותים, UPG), אך לעיתים זה המחיר ההכרחי של פרטיות.

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

לעומת:

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

נקודות האיזון היא הנפק החדשני שבו שני המסלולים שוויים בעלות.

גודל txetnoC wodniW

הו מספר הטוקנים המksiמלי שהמודל יכול לעבוד בשיחה אחת. זה כולל את כל ההיסטוריה של השיחה, את הפרופט, ואת התשובה הצפוייה.

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

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

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

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

למה זה חשוב? אם אתם צריכים לעבוד במסמכים ארוכים (חויזים, דוחות שנתיים, תיעוד טכני), גודל txetnoC wodniW יחסך את הצורך ב-RAG ובשימוש רב-שלבי.

משרד ערכיו דין צריך לנתח חוות חוות מורכבים באורך 0,05,000 מיליון. אם הם משתמשים ב-TPG 5.3-TPG, הם חייבים לפחות את החזזה לחלקם קטנים ולשלוח כל חלק בנפרד -- זה יוצר פיצול הקשר וסיכון להחמצת קשרים בין סעיפים.

פתרונות: שימוש edualC 5.3 : tennoS K002 עם txetnoC wodniW מאפשר לפחות את כל החזזה בחתה, וכך המודל רואה את התמונה המלאה.

עלות לפדי מודל (לחזזה בודד של K05 מילימ) = K76 טוקנים:

□ obruT 5.3-TPG : פיצול ל-5 קריאות □ \$ 200.0 / K1 = \$ 40 (אך אינטגרציה יותר)

□ edualC 5.3 : tennoS K002 קריאה אחת □ \$ 300.0 / K1 = \$ 02.0 (אינטגרציה יותר)

2.2.1 מדדי השוואה: tsoC sv ecnamrofreP

אחד הדרכים הטובות ביותר להשוות מודלים היא באמצעות tsoC-ot-ecnamrofreP,[9],[6] oitaR tsoCsv.[1] מציג השוואת מודלים המובילים, ואIOR מדגים את היחס בין ביצועים לעלות.

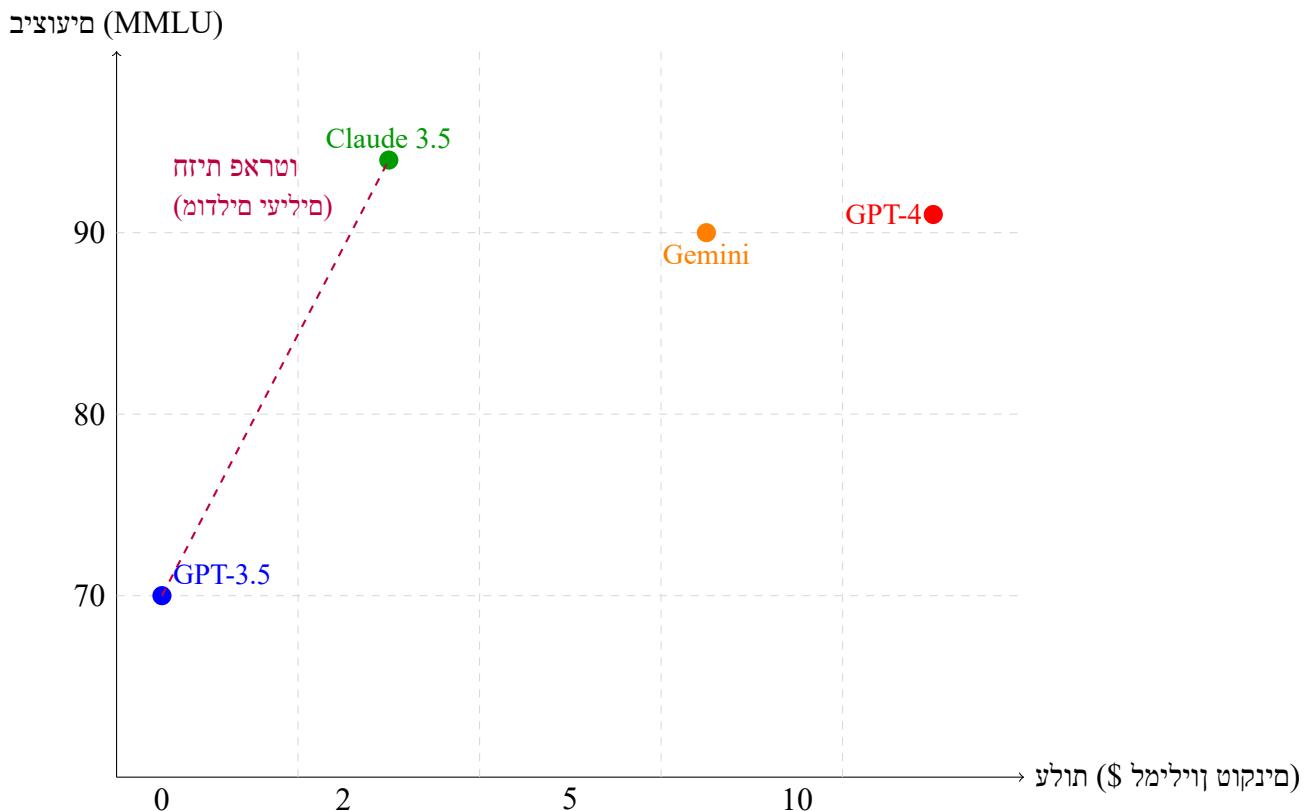
$$\text{Performance/Cost Ratio} = \frac{\text{Performance Score}}{\text{Monthly Cost}}$$

כasher:

- ציון מנורמל (0-100) ממדדים כמו MMLU, ULMM, lavEnamuH, או בדיקה פנימית
- עלות חודשית משוערת לפי נפח שימוש

טבלה 1.1: השוואת מודלים מוביילים (5202)

| לזום | MMLU | מחיר 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 |



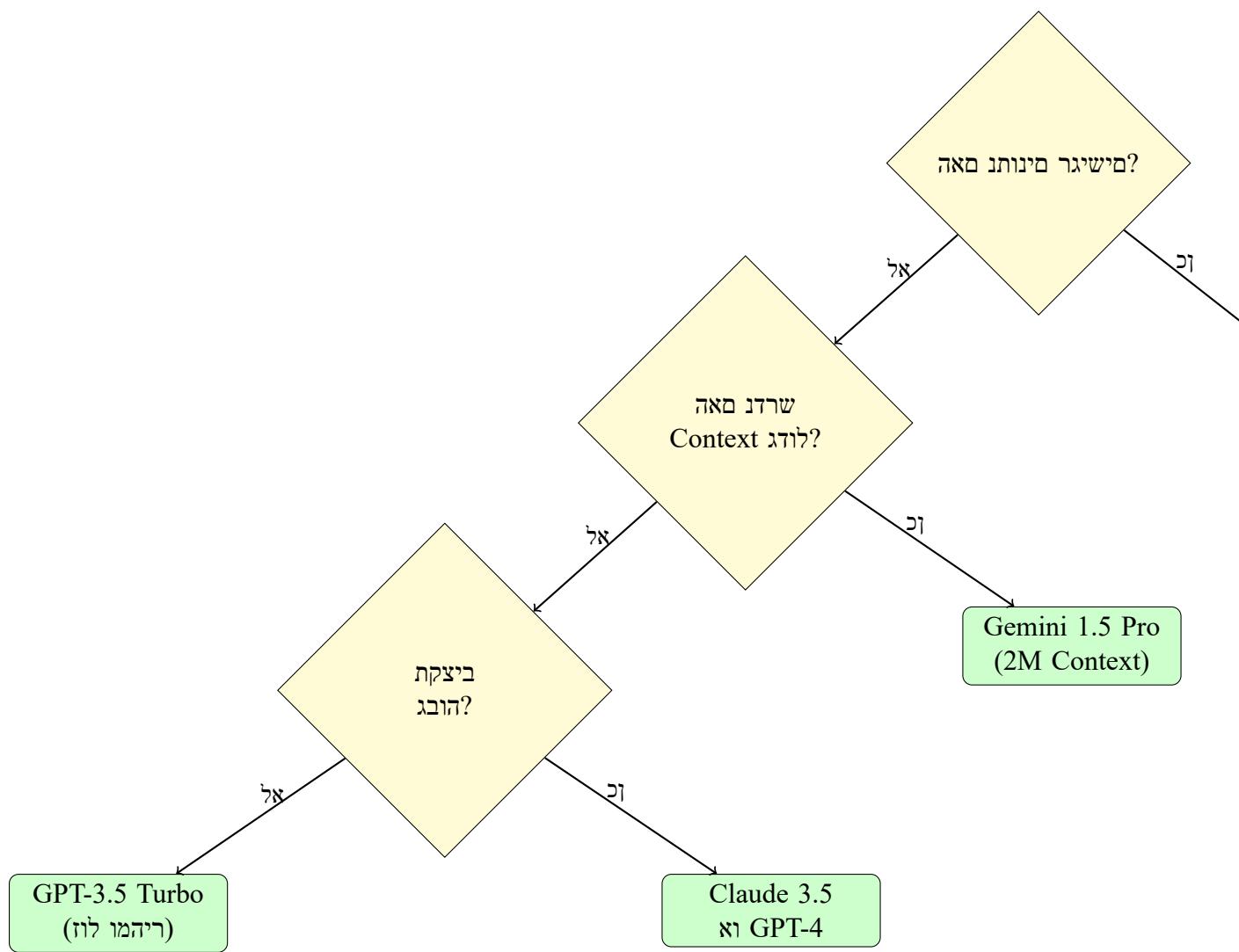
איור 1.1: גראף: ביצועים מול עלות

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

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

3.2.1 eerT noisiceD לבחירה MLL

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



איור 2.1: eerT noisiceD לבחירה MLL

3.1 בחירת ?lareneG cificepS-ksaT :sledoM gniddebmE או

המונע שמאחורי מערכות **RAG** וחיפוש סמנטי [31] [01]. הם הופכים טקסט לוקטורים מספוריים, מה שמאפשר למדוד "דמיון" בין משפטיים, מסמכים או שאלות.

1.3.1 sgniddebmE esopruP-lareneG

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

יתרונות:

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

חסרונות:

- פחות מדויק בתחוםים מאוד ספציפיים (רפואה, משפטיים, כימיה)
- עלויות IPA מתחשכות (אם משתמשים ב-IPA חיוני)

sgniddebmE cificepS-ksaT 2.3.1

במקרים שבהם הטקסט שלכם מאוד ספציפי -- למשל, מסמכים רפואיים, תקנות משפטיות, או קוד -- כדאי לש拷ול Fine-Tuning של מודל gnniddebmE או שימוש במודל ייעודי.

חברת תרופות גדולה השתמשה ב-**text-embedding-3-large** לאחיזור מאמרים מדעיים, אך גילתה שהבדיקה נזוכה -- המודל לא הבין טוב מונחים רפואיים כמו "ydicixotorhpen" או ".054P emorhcotyc".
הפתרון: enuT-eniF של מודל **BGE-M3** על 5,000,000 מאמרים רפואיים.
תוצאה:

- דיק אחזור עלה מ-56% ל-98%
- עלות חד-פעמית: \$0,500 (אימון)
- עלות שוטפת: \$0,002/חודש (detsoH-fleS)
- לעומת: \$0,200/חודש (IAnepO IPA) של OpeAnepI

נקודות איזון: 5.2 חודשים.

4.1 בחרית ?dirbyH ,lanoitaleR ,rotceV :esabataD

כאשר בונים מערכת AI, אחת השאלות הקritisיות היא: איך לאחסן את הנתונים?

sesabataD rotceV 1.4.1

[2] sgniddebmE כמו Qdrant ,Weaviate ,Pinecone Vector Databases [11]. הםאפשרים חיפוש לפי דמיון (Similarity Search) במהירות גבוהה.
מתי להשתמש:

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

sesabataD lanoitaleR 2.4.1

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

□ כאשר הנתונים מובנים (לקוחות, הזרנות, מלאי)

□ כאשר יש צורך ב-ACID Transactions (עסקאות אוטומטיות)

□ כאשר השאלות הן מדויקות (W, EREHW, NIOJ)

hcaorppA dirbyH 3.4.1

גישה היברידית משלבת את שני העולמות: נתונים מובנים ב-BD ונתונים מוצריים ב-RDBMS.-table 2.1 מסכמת את ההבדלים בין הגישות השונות.

אתר מסחר אלקטרוני רוצה להציג המלצות מותאמות אישית.

ארכיטקטורה היברידית:

□ פרטיא לckoות, הזרנות, מלאי (נתונים מובנים)

□ sgniddebmE של תיאורי מוצרים והיסטוריה גישה (חיפוש סמנטי)

תהליך המלצה:

1. שלוף מ-PQL: היסטוריה רכישות של הלוקה

2. המר את זה ל-EQL וחפש ב-PQL: מוצרים דומים

3. שלוף מ-PQL: פרטי המוצרים שנמצאו (מחיר, מלאי)

4. הצג ללקוח

table 2.1: השוואת סוגים

| וירטירק | Vector DB | Relational DB | Hybrid |
|------------------|------------------------------|------------------|----------------|
| יטנסיס שופיה SQL | ויצם | דמתן אל | ויצם |
| העטיה תובכרום | דמתן אל | ויצם | ויצם |
| תולע | תינוניב | הCOMMON | ההובג |
| (Similarity) | ההובג-תינוניב דואמם הריהם | הCOMMON תיטיא | ההובג הריהם |

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

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

1.5.1: ניהול השיחה הנוכחית yromeM mret-trohS

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 נוספת מוסיפה ממוצע של 100 טוקנים, ובשיחה יש 20 הודעות:

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

זה בכלל שההודעה ה-1 נשלחת 20 פעמים, ההודעה ה-2 נשלחת 91 פעמים, וכו'.

אסטרטגיות לחישוב:

□ `wodniW txetnoT noitacnurT`: חתוך הודעות ישנות כשהן חורגות מ-C

□ `:noitazirammuS`: סכム כל 01 הודעות לפסקה אחת

□ `:wodniW gnidilS`: שמור רק את ה-N הודעות האחרונות

2.5.1: זיכרון בין שיחות yromeM mret-gnoL

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

درיכים לישום mret-gnoL:

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

2. **שיחות קודמות**: שמור `sgniddebmE rotceV` של כל שיחה, וכשמתחליה שיחה חדשה -- אחז שיחות רלוונטיות

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

סטודנטים בונה סוכן תמיכה ללקוחות. הם רוצים שהסוכן "יזכור" שיחות קודמות.

ארכיטקטורה:

□ `customer`: טבלה `customers` -- שם, מייל, העדפות

□ `customer_id`: שיחת עם `sgniddebmE` `:BDamorhC`

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

תהליכי:

1. לך מתחבר עם `customer_id=123`

2. שאלתה ל-QSergtsoP: שלוו העדפות בסיסיות
 3. שאלתה ל-BDamorhC: מצא 3 שיחות רלוונטיות מה עבר (דמיון סמנטי לנושא הנוכחי)
 4. בנה P: " משתמש 321 הוא צמחוני, בעבר התלונן על אי חור במשלו. הנה שיחות קודמות: ..."
 5. שלח ל-MLL
- תוצאה: הלקוח מרגיש "מובן" ולא צריך לחזור על עצמו.

3.5.1 גישה לידע חיצוני: yromeM lanretxE

[1] yromeM lanretxE הוא היכולת של ה-MLL לגשת מידע שלא נמצא בתוך ה-C-Function Calling [3]. זה נעשה בדרך כלל דרך RAG או דרך IPA של MRC, PRE, מג אוויר

□: אছזר מסמכים רלוונטיים מ-VBD והזרק אותם לפרופט GAR

□: אפשר ל-MLL לקרוא לפונקציות חיצונית IPA של IPA, MRC, מג אוויר

6.1 wodniW txetnoC: מוגבלות וסטרטגיית התמודדות

הוֹא המגבלה הקשיה ביותר של MLL [8], [91]. אם השיחה שלך חורגת ממנה, המודל פשוט לא יכול לקבל את הקלט.

1.6.1 אסטרטגיית התמודדות עם wodniW txetnoC מוגבל

gniknuhC והזקה חוזרת

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

1. חALK את המסמך ל-01 חלקים
2. שלח כל חלק: "סכם את החלק זהה"
3. אסוסף 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 -- שומר רק את ה-01 הודעות האחרונות, ו"סכום" את השאר לפסקה אחת.

דוגמה:

- הודעות 1-05: סוכמו ל-"המשתמש שאל על מוצרים, הוא מעוניין בטלפונים"
 - הודעות 15-06: שמורות במלואן
- כך אתה "זכור" את העבר, אבל לא משלם עבור כל הטוקנים.

שימוש במודל עם CtxetnoW wodniW גדוֹל

הדרך פשוטה ביותר: עברו למודל עם Ctxtetno גודל יותר.

- אם אתה משתמש ב-*TPG* (K821) *obruT* 4-*TPG* (K61), עברו ל-*5.3-TPG*
 - אם אתה צריך יותר, עברו ל-*C-edual* (M2) (*inimeG* 5.1 orP) או (*tennoS* 5.3 K002)

הסיפור הנסתר: ni-kcoL rodneV 7.1

Vendor Lock-in הוא המצב שבו הארגון הופך תלוי לחלוtin בספק אחד, ומעבר לספק אחר כרוכז בעליות אדירות או בלתי אפשרי לחלוtin [7], [41].

1.7.1 איז נוצר rodneV-kcoL ni במערכות AI?

1. שימוש ב-IPA ייעודי של ספק מסוים (למשל, gpt-4-vision) שאינו
לו חלופה בספקים אחרים

אימון מודל ייודי ב- O_{ANeI} -- לא ניתן להסביר אותו ל- $A_{nT-eniF}$.

3. kcoL tamroF ataD sgniddebmE של העבר שפה ייְהוּדִית קשיה (למשל, sgniddebmE של IAneP0 erehoC sgniddebmE-L לא תואמים).

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

2.7.1 אסטרטגיות למניעת rodneV-kcoL-ni

שכבת הפשטה (reyaL noitcartsbA)

במקום לקרוא יישורות ל- `openai.ChatCompletion.create`, בנה משק כללי שיכול לקרוא לכל ספק [4, 61].

Listing 1.1: Python: מילודם הטשפה תבכש

```
1 class LLMPProvider:
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
```

פרק 1. שיקולים אסטרטגיים – בחירת טכנולוגיות וניהול זיכרון

```
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("זהם AI?")
```

כעת, אם תרצה לעבור מ-OpenAI ל-Antheon, פשוט תשנה את הparameter provider -- לא צורך בשינוי קוד נוסף.

ygetartS ledoM-itluM

במוקום להתחביב למודל אחד, השתמש ב-**ygetartS ledoM-itluM**

□ **משימות קלות:** obruT 5.3-TPG

□ **משימות מורכבות:** tennos 5.3 edualC

□ **משימות רגניות:** (detsoH-fleS) 1.3 amall

כך אתה לא תלוי בספק אחד, וגם מפזר סיכוןים.

תיעוד ומבחן noissergeR

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

Listing 1.2: Python: Regression מילודים ו-חומר

```
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 סטטוס

לפני שמחלייטים לעבר ספק, חשוב לחשב את עלות המעבר:

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

כארה:

□: עלות פיתוח מחדש (שעות מהנדס □ שכר שעתי)

□ **C_{data}**: עלות העברת נתונים והמרת

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

□ **הדרכת הצוות על הכלי החדש**: C_{training}

חישוב: חברה משתמשת ב- $O(n^4)$ ורוצה לעבור ל- $O(n^3)$.

■ **פיתוח חדש:** 3 מהנדסים ■ 2 שבועות ■ 08 שעות ■ 001\$/שעה = 000,84\$

◻ **העברות הנוכחיים:** 000,01\$ = BG/20.0\$ ◻ sgniddebmE BG005
gniddebmE אחר(gniddebmE

000,51\$/יום הכנסות = 3 ימים 000,5\$/יום הכנסות = 3 ימים

הדרמה: 02 עובדים □ 4 שעות □ \$08/שעה =

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

החלטת: אם המעבר ל-IC edual \$3,000/חודש, נקודת האיזון היא 5.62 חודשים. האם זה כדאי? תלוי באסטרטגיית אורך התווחה.

8.1 דוגמאות מעשיות

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

תרחיש: חברת SaaS השתמשה ב-TPG 5.3-obruT לשיכום פניות לקוחות. הם קיבלו תלונות על דיקון נזוק -- המודול "החמיין" נקודות חשובות.

שיקולים:

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

□ **עלות:** יקר פי 02 (!)

□ **נפח:** 000,01 פניות/חודש, ממוצע 005 טווקנים לפניה

חישוב עלויות:

□ TPG 5.3-obruT : $10,000 \times 0.5K \times \$0.002 = \$10$

□ TPG 4 : $10,000 \times 0.5K \times \$0.03 = \$150$

החלטה: החברה עברה ל-TPG 4.

□ 80% פניות "רגילות": TPG 5.3-obruT (\$8/חודש)

□ 20% פניות "מורכבות": TPG 4 (\$03/חודש)

עלות כוללת: \$83/חודש -- שיפור אינטראקטיביות 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 או TPG 4?

בדיקות COP: הם הרכזו 001 שיחות אמיתיות במקביל על שני המודלים.

תוצאות:

| דגם | Claude 3.5 | GPT-4 |
|--------------------|------------|--------|
| תובושת קויד | 87% | 84% |
| עצומם הבוגת ומז | 2.3s | 4.1s |
| החיש/תולע | \$0.015 | \$0.08 |
| תווחקל ווצר תועיבש | 4.2/5 | 4.0/5 |

ההחלטה: edualC מהיר יותר, זול יותר, ודיקון טוב יותר. TPG 4 לא הוכיח את העלות.

3.8.1 דוגמה 3: תכנון אסטרטגי ledoM-itluM

תרחיש: ארגון גדול רוצה לבנות מערכת IA שמתפלת במגוון משימות.

ארכיטקטורה:

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

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

□ **נתונים רגיסטים:** B07 1.3 amalL detsoH-fleS (פרטיות מלאה)

□ **יצירת קוד:** 4-TPG (מצטין בקוד)

תוצאה:

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

□ איקות: גבולה בכל תחום

□ גישות: אפשר להחליף ספק בכל תחום בנפרד

9.1 תרגילים

1.9.1 תרגיל תיאורטי 1: בניית draceroc להשוואת sMLL

משימה: בנה draceroc להשוואה 4 מודלים: B07 1.3 amalL ,orP 5.1 inimeG ,5.3 edualC ,4-TPG קרייטריוניים:

□ ביצועים (ULMM)

□ עלות למיילון טוקן

wodniW txetnoC □

ycnetaL □

□ רגישות נתונים (אם ?detsoH-fleS

תן ציון 1-10 לכל קרייטריון, וشكلל לפי חשיבות לארגון שלך.
פתרונות לדוגמה:

| תק | GPT-4 | Claude 3.5 | Gemini | Llama 3.1 |
|-------------------|------------|------------|------------|------------|
| (30%) סייעוציב | 9 | 10 | 8 | 7 |
| (25%) תולע | 3 | 7 | 5 | 10 |
| Context (20%) | 7 | 9 | 10 | 7 |
| Latency (15%) | 5 | 7 | 4 | 8 |
| (10%) תוטרפ | 2 | 2 | 2 | 10 |
| ללוב וויאץ | 6.3 | 8.0 | 6.7 | 8.0 |

המלצת: 1.3 amalL או -- תלוי אם אתה מוכן לנחל .detsoH-fleS 5.3 edualC

2.9.1 תרגיל תיאורטי 2: חישוב gnihctiwS tsoC

תרחיש: חברת משתמשת ב-P(BD) enoceniP tnardQ rotceV ורוצה לעבור ל-Q(rotceV).
נתונים:

□ 01 מיליון enoceniP ב-sgniddebmE

□ עלות P/005\$: enoceniP

□ עלות QtnardS-fleH-detS (\$002/חדש (שרות)

□ זמן העברת משוער: 3 שבועות (2 מהנדסים)

□ שכר מהנדס: \$021/שעה

חשב:

tsoC gnihctiwS .1

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

החלטה: לא כדאי -- tsoC gnihctiwS גבוה מדי.

3.9.1 תרגיל תיאורטי 3: תכנון אסטרטגיית tnemeganaM txetnoC

תרחיש: אתה בונה סוכן שיחה ללקוחות. שיחת ממוצעת היא 30 דקות, כל הודעה 051 טוקנים.

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

משימה: תכנן 3 אסטרטגיות tnemeganaM txetnoC וחשב את העלות של כל אחת.

פתרון:

tsoC gnidilS .1

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

□ עלות: $69,750 \times \$0.002 / 1000 = \0.14 לשיחה

wodniW gnidilS .2

□ טוקנים לשיחה: בממוצע $150 \times 10 \times 30 = 45,000$

□ עלות: $45,000 \times \$0.002 / 1000 = \0.09 לשיחה

noitazirammus .3

□ טוקנים לשיחה: סכום כל 500 (summary) + 1,500 (last 10) = 2,000

□ עלות: $2,000 \times \$0.002 / 1000 = \0.004 לשיחה

המלצת: -- חיסכון של 79%!

4.9.1 תרגיל תיאורטי 4: ניתוח ni-kcoL rodneV

תறיש: בדוק את הארכיטקטורה הבאה זהה נקודות V :ni-kcoL rodneV

▪ שולח בקשות שירות ל-O IPA IAneF dnetnorF

▪ משתמש ב-002 text-embedding-ada-002 sgniddebmE

▪ שמור ב-P (ענן) enoceniP esabataD

▪ פתרון: ft:gpt-3.5-turbo:company:v1 :ledoM denuT-eniF

משימה:

1. זהה 4 נקודות ni-kcoL

2. הצע פתרון לכל נקודה

פתרונות:

1. ni-kcoL #1: קורא שירות ל-O IPA dnetnorF

▪ פתרון: הוסף dnekcaB reyaL noitcartsbA

2. ni-kcoL #2: ייעודיים ל-O IPA sgniddebmE

▪ פתרון: השתמש במודל gniddebmE נייטרלי (detsoH-fleS 3M-EGB)

3. ni-kcoL #3: בען enoceniP

▪ פתרון: עבר ל-Q etaivaeW או tnardQ ייעודי

4. ni-kcoL #4: מודל אחר ledoM denuT-eniF

▪ פתרון: שומר את ataD gniniarT -- אפשר לאמן מחדש על מודל אחר

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

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

שלב 1 (שנה 1):

▪ COP עם 4-TPG IPA (מהיר לישום)

▪ BD rotceV בען (enoceniP)

▪ 1,000 לקוחות

שלב 2 (שנה 2):

▪ מעבר ל-C + 4-TPG ledoM-itluM

▪ הוספה detsoH-fleS (3M-EGB) ledoM gniddebmE

▪ 0,000,01 לקוחות

שלב 3 (שנה 3):

▪ MLL detsoH-fleS (4 amall) לקבלת נתונים רגילים

▪ BD rotceV detsoH-fleS (tnardQ)

▪ 0,000,001 לקוחות

תוצאה: גמישות מלאה, עליות 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: פילוגם יוציב התואשה

```

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__}: {e}")
87
88         self.print_summary()
89
90     def print_summary(self):
91         print("\n==== SUMMARY ===")
92         models = set([r['model'] for r in self.results])
93
94         for model in models:
95             model_results = [r for r in self.results if r['model'] == model]

```

```

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

```

תוצאה צפואה:

==== 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 שמנהל זיכרון שיחה ארוכה באמצעות סיכומים אוטומטיים.

Listing 1.4: Python: מוכיס מע וורכיז לוהין

```

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 // 2]
24
25        # Create summary prompt
26        conversation_text = "\n".join([
27            f"{msg['role']}: {msg['content']}"
28            for msg in messages_to_summarize
29        ])
30
31        summary_prompt = f"""Summarize this conversation in 2-3
sentences:
{conversation_text}
32
33 Previous summary: {self.summary if self.summary else 'None'}
34
35 Provide a concise summary that captures key points."""
36
37
38    # Generate summary
39    response = openai.ChatCompletion.create(
40        model="gpt-3.5-turbo",
41        messages=[{"role": "user", "content": summary_prompt}],
42        max_tokens=200
43    )
44
45
46    new_summary = response.choices[0].message.content
47
48    # Update summary
49    if self.summary:
50        self.summary = f"{self.summary}\n\n{new_summary}"
51    else:
52        self.summary = new_summary
53

```

פרק 1. שיקולים אסטרטגיים – בחירת טכנולוגיות וניהול זיכרון

```
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
58     def get_context_for_llm(self) -> List[Dict]:
59         """Get full context to send to LLM"""
60         context = []
61
62         # Add summary as system message if exists
63         if self.summary:
64             context.append({
65                 "role": "system",
66                 "content": f"Previous conversation summary:\n{self.
67 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"\n 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 הודיעות האחרונות

.2. כשעוברים את המגבלה, היא מסכמת את המחזית הראשונה

.3. הסיכום נשלח כ-Summary מטעס egasseM בכל קרייה

4. כך חוסכים טוונטים אבל שומרים על הקשר

01.1 סיכום

בפרק זה למדנו כיצד לקבל החלטות אסטרטגיות בעולם ה-IA העסקי [5]. ראיינו שבבחירה **LLM** אינה רק שאלת טכנית -- היא החלטה עסקית שמשפיעה על עלויות, ביצועים, פרטיות וגישות עתידית.

נקודות מפתח:

- **אין מודל "מושלם"** -- כל מודל מצטיין במשימות מסוימות. ledoM-itluM ygetartS מושלם מושלם. קרוביות הפתרון הטוב ביותר.
 - **wodniW txetnoC הוא מגבלת קרייטית** -- תכנן מראש איך אתה מנהל זיכרון, ובחר מודל עם txetnoC מותאים למשימות שלך.
 - **V rodneV kcoL txetnoC הוא סיכון אמיתי** -- השקיעו בשכבות הפשטה ובארכיטקטורה גמישה מהיום הראשון.
 - **חשיבות אrox טוח tsoC gnihctiwS** יכול להיות עצום. תכננו pamdaoS טכנולוגי ל-3 שנים לפחות.
- בפרק הבא נעסוק בממשק משתמש וинтерאקטיבי -- איך להפוך את כל הטכנולוגיה הזאת לחוויה משתמש מעולה.