

הנדסת פרומפטים: מתמטיקה, שיטה וביצוע למערכות AI מתקדמות

Prompt Engineering: Mathematics, Method, and Execution
for Advanced AI Systems

ד"ר יורם סגל

כל הזכויות שמורות - © Dr. Segal Yoram

December 2025

גרסה 1.0

תוכן העניינים

6	1 סודות תיאורתיים של הנדסת פרומפטים	1
6	הגדרות יסוד: Fundamental Definitions	1.1
7	מבוא: המהפכה השקטה של השפה	1.2
7	הפרדוקס של העדרויות: The Paradox of Absences	1.3
7	משפט Wiio למודלי שפה גדולים	1.4
8	מרחב ההסתברות המותנית: Conditional Probability Space	1.5
8	יסודות מתמטיים: Mathematical Foundations	1.6
9	איך מחשבים את H – אי-זדות התשובה בהינתן הפרומפט	1.6.1
9	דוגמה מספרית: פרומפט טוב מול פרומפט גרוע	1.6.2
10	המסקנה המעשית	1.6.3
11	מורכבות (Perplexity): כמה המודל מופתע?	1.6.4
11	מבחן הניחוש: אינטואיציה פשוטה	1.6.5
12	איך מחשבים מורכבות? שלושה צעדים	1.6.6
12	דוגמה מספרית: "החתול ישב"	1.6.7
12	השוואה: משפט הגינוי מול משפט מוזר	1.6.8
13	הקשר למשוואה 1	1.6.9
14	השוואת מטרות פרומפטים: Prompt Objectives Comparison	1.7
16	ויזואיזציה של זרימת מידע: Information Flow Visualization	1.8
16	דוגמאות עשה ועל תעשה: Do and Don't Examples	1.9
18	סיכום הפרק: Chapter Summary	1.10
19	2 מתודולוגיה מערכית לתכנון פרומפטים	2
19	עקרון הפרומפט האוטומי: Atomic Prompting Principle	2.1
20	פירוש הטבלה: ארבעה שלבים בתהליכי העבודה	2.1.1
21	השוואת פרומפטים: Prompt Differing	2.2
22	כל שלושת הפרומפטים: The 3-Prompt Rule	2.3
23	נוסחאות מתמטיות: Mathematical Formulas	2.4
24	דוגמאות עשה ועל תעשה: Do and Don't Examples	2.5
26	סיכום הפרק: Chapter Summary	2.6
27	3 תבניות פרומפט מתקדמות וארכיטקטורות	3
27	מבוא: תפקיד המתכנת בעידן ההסקה	3.1
27	הFIGOM החיצוני	3.1.1
27	שני מישורים של בקרה	3.1.2
28	המסקנה החשובה	3.1.3
28	שרשרת מחשבה: Chain-of-Thought	3.2
28	הגדרה פורמלית	3.2.1

28	הסבר נרטיבי	3.2.2
29	דוגמה פשוטה	3.2.3
29	גרסה מתקדמת: Chain-of-Thought++	3.2.4
29	מהות השיטה המתקדמת	3.2.5
30	הצבעת עקביות עצמית	3.2.6
30	: שילוב חשיבה ופעולה ReAct	3.3
30	הגדרה פורמלית	3.3.1
30	הסבר נרטיבי	3.3.2
31	דוגמה פשוטה	3.3.3
31	גרסה מתקדמת: ReAct 2.0	3.3.4
31	מעבר לפראדיגמה חדשה	3.3.5
31	ארכיטקטורת הזרימה	3.3.6
33	: עץ מחשבות Tree-of-Thoughts	3.4
33	הגדרה פורמלית	3.4.1
33	הסבר נרטיבי	3.4.2
33	דוגמה פשוטה	3.4.3
33	גרסה מתקדמת: Tree-of-Thoughts עם תקציב כפוי	3.4.4
34	חקירה מבנית	3.4.5
34	גיזום דינמי	3.4.6
34	הנחיתת תפקידים: Role-Based Prompting	3.5
36	למידה מדוגמאות: אופטימיזציה של Few-Shot	3.6
36	השווואה מערכתי בין תבניות	3.7
36	דוגמאות עשה ועל תעשה: Do and Don't Examples	3.8
38	חלון ההקשר בפעולה: Context Window Mechanics	3.9
38	: צמיחה החלון Chain-of-Thought	3.9.1
39	: עצירות וחזרות ReAct	3.9.2
39	: ענפים מקבילים Tree-of-Thoughts	3.9.3
40	מה קורה כשהחלון נגמר?	3.9.4
40	סיכום הפרק: Chapter Summary	3.10
41	הנדסת פרומפטים לסוכני IA	4
41	מבוא: מהמענה לפעולה	4.1
41	משמעות סוכן-מחשב: (ACI)	4.2
42	תבנית ReAct	4.2.1
42	ארכיטקטורות סוכנים: Agent Architectures	4.3
43	מעקות בטיחות כפרומפטים: Guardrails as Prompts	4.4
43	מטא-פרומפטינג: Meta-Prompting	4.5
44	נוסחאות מתמטיות: Mathematical Formulas	4.6
45	שימוש בכלים: Tool-Use	4.7

45	דוגמאות עשה ולא תעשה: Do and Don't Examples	4.8
46	סיכום הפרק: Chapter Summary	4.9
47	הערכה, אופטימיזציה והטמעה ביצור	5
47	מבוא: מההעבדה לייצור	5.1
47	ניהול גרסאות פרומפטים: Prompt Versioning	5.2
48	mobichkot statystichit: Statistical Significance	5.3
48	metodologiyat b'dikot A/B: A/B Testing Methodology	5.4
49	zihovi schifah: Drift Detection	5.5
50	zinor CI/CD l'perfomptim: CI/CD Pipeline	5.6
51	opfotimiziyat ulot-dik: Cost-Accuracy Optimization	5.7
52	dogmavot usha ve'al tasha: Do and Don't Examples	5.8
53	simcos hafrak: Chapter Summary	5.9
54	mekorot be'ivrit: Makorot B'Hebrew	5.10
5.11 English References		54

רשימת האיורים

16	זרימת מידע דרך שכבות Attention בטרנספורמר	1
17	משתח דיק כפונקציה של אורך פרומפט וטמפרטורה (מערך נתונים GSM8K)	2
17	השוואה בין פרומפטים אוטומטיים לפרומפט מונוליטי. הגישה האוטומטית	3
20	משימה דיק כולל של 89% לעומת 72% בגישה המונוליטית.
23	שיפור איטרטיבי: דיק עולה והזיות יורדות לאורך ארבע איטרציות של שיפור פרומפט.	4
24	מחזור החיים של הנדסת פרומפטים: מהгадרת משימה ועד פרישה ביצור.	5
32	ארכיטקטורת ReAct 2.0: מעגל Act-Observe-Reflect-Adapt	6
35	ען מחשבות עם גיאום: ענפים עם ציון נמוך נגזרים	7
50	תוצאות ניסוי A/B: השוואת גרסאות פרומפט לאורך 7 ימים	8
51	zinor CI/CD l'perfomptim: משלב הפיתוח ועד לייצור	9

רשימת הטבלאות

15	השוואה מטרות פרומפטים לפי מדדי מידע	1
15	מדדי אופטימליות פרומפט לדגמים שונים (2025)	2
20	פירוק פרומפט לשימוש ניתוח קוד	3
22	מדדי שיפור איטרצייה-אחר-איטרציה	4
36	אופטימיזציה few-shot learning	5
37	השוואה תבניות פרומפטים לפי סיבוכיות	6
38	חלון הקשר ב-CoT - שלב 1	7

38	חולון הקשר ב-CoT – שלב 2	8
39	חולון הקשר ב-ReAct – מהזור שלם	9
39	חולון הקשר ב-ToT – הערכת ענפים	10
42	ארכיטקטורות פרומפטים לSOCNNIM	11
43	סוגי מעוקות בתייחות ופרומפטים	12
49	מתודולוגיות בדיקות A/B לפרומפטים	13
52	מדדי הערכתה לפרומפטים בייצור	14

1 יסודות תיאורטיים של הנדסת פרומפטים

מטרות הלמידה

בסיום פרק זה, הקורא יוכל:

- להסביר את מנגנון ה-Attention ואת הקשר שלו ליעילות פרומפטים
- לבנות מודל מידע לתכנון אופטימלי של tokens
- להגדיר פורמלית את מרחב הפרומפטים ופונקציית התשובה
- לנתח את הפרדוקס של העדרויות בתכנון פרומפטים

1.1 הגדרות יסוד: Fundamental Definitions

לפני שנצלול לעומק, נגדיר את המושגים המרכזיים.

מהו פרומפט?

המילה prompt באנגלית פירושה "הנחיה" או "זירוז לפעולה". בעברית, המונח המקביל הוא **הנחיה**. לאורך הספר נשתמש בשני המונחים לסירוגין: **הנחיה** ו-**פרומפט** (בכתב עברי).

הגדרה: פרומפט (Prompt)

הגדרה פורמלית: פרומפט הוא רצף של טקסט בשפה טבעית המשמש כקלט למודל שפה גדול (LLM), במטרה להנחות את המודל לייצר פלט רצוי.

במילים פשוטות: פרומפט הוא השאלה שאתה שואל את המכונה. אבל זה יותר מאשר. זו הדרך שבה אתה מסביר למוכונה מה אתה רוצה. פרומפט טוב הוא כמו הוראות ברורות לעובד חדש. פרומפט גרוע הוא כמו לבקש "תעשה משהו טוב".

מהי הנדסת פרומפטים?

הגדרה: הנדסת פרומפטים (Prompt Engineering)

הגדרה פורמלית: הנדסת פרומפטים היא התהlik השיטתי של עיצוב, מבנה ואופטימיזציה של הנחיות טקסטואליות, במטרה לשפר את יכולות הפלט ממודלי בינה מלאכותית גנרטיבית [1].

הגדרה זו מבוססת על סקירה אקדמית מקיפה מ-2024 שנייה למעלה מ-1500 מאמרים בתחום.

במילים פשוטות: הנדסת פרומפטים היא האמננות וההduced של לדבר עם מכונות. זה לא רק מה שאתה אומר. זה איך אתה אומר. באיזה סדר. עם כמה פרטיים. מהנדס פרומפטים טוב מבין את המכונה. הוא ידע מה היא צריכה לשמעו כדי לעבוד נכון.

למה זה חשוב?

חישבו על זה כך: יש לכם כלי עצמאי. המפתח לשימוש בו הוא התקשורת. מודלי שפה גדולים יכולים לכתוב קוד, לנתח נתונים, ליצור תוכן. אבל הם עושים רק מה שמקשים מהם. והבקשה היא הפרומפט.

לכן הנדסת פרומפטים היא לא רק מיומנות טכנית. היא מיומנות תקשורת. היא דורשת חשיבה ברורה, ניסוח מדויק, ומבנה של הצד השני. בספר זה נלמד את שנייהם: את התיאוריה המתמטית ואת האמננות המעשית.

1.2 מבוא: המהפכה השקטה של השפה

לפני 70000 שנה קרה משהו מוזר. בני אדם התחליו לדבר. השפה שנינה הכל. היא אפשרה לנו לשתף רעיונות. היא יקרה תרבויות שלמות. היום אנחנו עומדים בפני מהפכה דומה. מודלי שפה גדולים (Large Language Models) הם הישג טכנולוגי עצום. הם מבינים שפה אנושית. הם יוצרים טקסט חדש. אבל יש בעיה: איך אנחנו מתקשרים איתם? התשובה היא **הנדסת פרומפטים** [1]. פרומפט הוא הוראה בשפה טבעית. הוא אומר לנו מה לעשות. פרומפט טוב מניב תוצאות טובות. פרומפט גרוע מניב שטויות. הפרק הזה מציג את היסודות התיאורתיים של התחומים. נלמד מתמטיקה. נבין מדוע פרומפטים עובדים. נגלה את הכללים שמנחים תקשורת עילית עם מכונות.

1.3 הפרדוקס של העדרויות: The Paradox of Absences

הפרדוקס של העדרויות

פרומפט ממוקד מדי מפספס פתרונות יצירתיים; פרומפט רחב מדי מגביר hallucinations. האתגר הוא למצוא את נקודת האיזון האופטימלית.

כשאנו כותבים פרומפט, אנחנו עומדים לפני דילמה. אם הפרומפט ספציפי מדי, המודל מוגבל. הוא לא יכול לחשב מהו לקופה. אם הפרומפט כללי מדי, המודל מבולבל. הוא עלול להמציא עובדות [2].
מחקרנים מראים שהzieות (Hallucinations) קורות ב-5 עד 15 אחוז מההתשובות. הסיבה: המודל מלא חסר. כשהאין מספיק מידע, הוא מנחש [3].
הפתרון הוא איזון. פרומפט טוב נותן מספיק הקשר. לא יותר מדי. לא פחות מדי. זה הפרדוקס של העדרויות: מה שלא נכתב חשוב כמו מה שנכתב.

1.4 משפט Wiio למודלי שפה גדולים

משפט Wiio ל-LLMs

Communication usually fails, except by accident." כל פרומפט דורך validation דרך דרך גבולות תיאורתיים-מידיים.

פרופסור אוסמו וייאו הציג חוק פשוט: תקשורת בדרך כלל נכשלת. זה נכון גם למודלי שפה. רוב הפרומפטים לא עובדים כמו שציפינו. הסיבה נעוצה בתורת המידע [4]. כל הודעה מכילה מידע. המידע נמדד ב-bits. כשהמידע בפרומפט לא מספיק, המודל מלא חסר. כשהמידע מרובה מדי, המודל מתבלבל. האנטרופיה (Entropy) מוגדרת אי-זדאות. פרומפט טוב מצמצם אי-זדאות. הוא מכוען את המודל לתשובה הנכונה. פרומפט גורע מגביר אי-זדאות. הוא משאיר את המודל מבולבל.

1.5 מרחב הסתברות המותנית: Conditional Probability Space

מרחב הסתברות המותנית

קיימת נקודת אופטימום שקשרורה $P(\text{answer}|\text{prompt})$ איננו מונוטוני ב-context length. למורכבות perplexity של המודל.

מודלי שפה הם מכונות הסתברותיות. הם מחשבים: מה הסיכוי לתשובה y בהינתן פרומפט x ? זו הסתברות המותנית: $P(y|x)$. אבל יש הפתעה. פרומפט ארוך יותר לא תמיד טוב יותר. מחקר חשוב מ-2024 גילתה תופעה מעניינת [5]. המודלים שוכחים מידע באמצעות הפרומפט. הם זוררים את ההתחלה והסוף.

זה נקרא "Lost in the Middle". אורך הפרומפט משפיע על הדיוק. אבל לא בצורהلينיארית. יש נקודת אופטימלית. פרומפט קצר מדי חסר מידע. פרומפט ארוך מדי מבולבל. איור 2 מציג את הקשר בין אורך, טמפרטורה ודיק. אפשר לראות את ה-ridge האופטימלי: עד 300 טוקנים בטמפרטורה 0.7.

1.6 יסודות מתמטיים: Mathematical Foundations

עשוי נפרמל את הרעיון. נגדיר משוואות שמתארות פרומפטים אופטימליים.

משואה 1.1: פונקציית אובדן לפרומפטים

$$(1) \quad \mathcal{L}_{\text{prompt}} = \alpha \cdot H(y|x) + \beta \cdot \frac{|x|}{C_{\max}} + \gamma \cdot \text{Perplexity}(x)$$

המשואה מגדירה פונקציית אובדן [6]. ככל שהערך נזוק יותר, הפרומפט טוב יותר.

- x = הפרומפט שכתבנו

- y = התשובה שאנו רוצים

- $H(y|x)$ = אי-ודאות התשובה בהינתן הפרומפט

- C_{max} = גודל חלון ההקשר של המודל

- α, β, γ = משקלות שקובעים את האיזון

הרכיב הראשון מודד איזוט. הרכיב השני מודד יעילות. הרכיב השלישי מודד בהירות.

1.6.1 איך מחשבים את H – אי-ודאות התשובה בהינתן הפרומפט

עמדו רגע. ראיינו את המשטנה $H(y|x)$ במשוואה. אבל מה זה אומר בפועל? איך מחשבים "אי-ודאות"? התשובה מגיעה מטורת המידע של שאנו [4].
כשאנו שואלים מודל שפה שאלת, הוא לא באמת "יודע" את התשובה. הוא מחשב הסתברויות. לכל מילה אפשרית הוא נותן ציון בין 0 ל-1. המילה עם הציון הגבוה ביותר נבחרת כתשובה.
אבל מה קורה כשהמודל לא בטוח? כשהכמה מילים מקבלות ציונים דומים? זה בדיק מה שאנטרופיה מוגדרת: כמה המודל "מבולבל".

נוסחת האנתרופיה המותנית

$$(2) H(Y|x) = - \sum_{i=1}^n P(y_i|x) \cdot \log_2(P(y_i|x))$$

מקרה משתנים:

- n = מספר המילים האפשריות באוצר המילים של המודל

- y_i = מילה ספציפית (למשל: "עברית", "אנגלית", "צרפתית")

- $P(y_i|x)$ = ההסתברות שהמודל נותן למילה y_i בהינתן הפרומפט x

- \log_2 = לוגריתם בסיס 2 (התוצאה נמדדת בבייטים)

נשמע מסובך? בוואו נראה דוגמה פשוטה.

1.6.2 דוגמה מספרית: פרומפט טוב מול פרומפט גרוע

נניח שאנו מזינים מודל את הפרומפט: "...In Israel, people speak the...". המודל צריך לחזות את המילה הבאה.

תרחיש א': פרומפט מוקד (מודל בטוח)

הפרומפט ברור. המודל "יודע" מה התשובה. הוא נותן את ההסתברויות הבאות:

- "Hebrew": 0.9

0.1 :"Arabic" -

- כל שאר המילים: 0

נחשב את האנתרופיה:

$$\begin{aligned} H &= -[0.9 \cdot \log_2(0.9) + 0.1 \cdot \log_2(0.1)] \\ &= -[0.9 \cdot (-0.152) + 0.1 \cdot (-3.321)] \\ &= -[-0.137 + (-0.332)] \\ &= \mathbf{0.47} \text{ bits} \end{aligned}$$

תוצאה טובה

ערך אנתרופיה נמוך (0.47 ביטים) = המודל בטוח בתשובה שלו. זה מה שאנו רוצים. הפרומפט עובד.

תרחיש ב': פרומפט עמוס (מודל מבולבל)

עכשו נשנה את הפרומפט ל: "...The language is...". אין הקשר. על איזו מדינה מדובר? המודל לא ידע.

ההסתברויות מתפוזרות:

0.5 :"English" -

0.5 :"Hebrew" -

נחשב:

$$\begin{aligned} H &= -[0.5 \cdot \log_2(0.5) + 0.5 \cdot \log_2(0.5)] \\ &= -[0.5 \cdot (-1) + 0.5 \cdot (-1)] \\ &= -[-0.5 + (-0.5)] \\ &= \mathbf{1.0} \text{ bit} \end{aligned}$$

▲ שימוש לב להבדל

ערך אנתרופיה גבוהה (1.0 ביט) = המודל מבולבל. הפרומפט לא עובד. צריך לשפר אותו.

1.6.3 המסקנה המעשית

חזרו למשואה 1. המטרה היא **למצער את $H(y|x)$** . הרכיב \mathcal{L}_{prompt} הוא חלק מהמשוואות. ככל שהוא קטן יותר, הפרומפט טוב יותר.
איך מקטינים את H ? בדיק כמו בדוגמה:

- פרומפט ברור ומוקד → המודל בטוח $\rightarrow H$ נמוך

- פרומפט עמוס וכלי → המודל מבולבל $\rightarrow H$ גבוהה

במילים פשוטות: $H(y|x)$ הוא **מדד המבוכה** של המודל. כשהמודל "יודע" מה לענות, המבוכה נמוך. כשהוא מוחש, המבוכה גבוהה. תפקיד מהנדס הפרומפטים הוא להפחית את המבוכה לMINIMUM.

1.6.4 מרכיבות (Perplexity): כמה המודל מופתע?

עכשו נכיר מدد נוסף: **רכיבות** (באנגלית: Perplexity). המילה המקורית פירושה "מבוכה" או "תימהון". זה בדוק מה שהמדד מודד.
חשבו על זה כך: האנתרופיה H מודדת מבוכה בBITS. המרכיבות מודדת את אותו מבוכה במונחים אחרים: **מספר האפשרויות הסבירות**.

1.6.5 מבחן הניחוש: אינטואיציה פשוטה

דמיינו שאתם מנסים לנחש את המילה הבאה במשפט.

- אם המרכיבות היא 1: אתם בטוחים לחולוטין. יש רק אפשרות אחת הגיונית.
 - אם המרכיבות היא 6: אתם מבולבלים כאילו הייתם צריכים להטיל קובייה.
 - אם המרכיבות היא 100: אתם אבודים לחולוטין. כמו לבחור קלף מחפיסה כפולה.
- כל שהמספר נמוך יותר, המודל פחות מופתע. ככל שהמספר גבוה יותר, המודל בהלים.

נוסחת המרכיבות (Perplexity)

$$(3) \quad PP(x) = \sqrt[N]{\frac{1}{P(w_1) \cdot P(w_2) \cdot \dots \cdot P(w_N)}}$$

מקרה משתנים:

$x = PP(x)$ - המרכיבות של הטקסט x

$N =$ מספר המילים בטקסט

$w_1, w_2, \dots, w_N =$ המילים בטקסט לפי הסדר

$P(w_i) =$ ההסתברות של המילה w_i בהינתן המילים שלפנייה

$\sqrt[N]{\dots} =$ שורש מסדר N (ממוצע הנדסי)

נשמע מסובך? בואו נפרק את זה לשלבים פשוטים.

1.6.6 איך מחשבים מורכבות? שלושה צעדים

צעד 1: מוצאים הסתברות לכל מילה. מה הסיכוי שהמילה הזו תופיע, בהינתן המילים שבאו לפניה?

צעד 2: מכפילים את כל הסתברויות. מקבלים את הסיכוי לכל המשפט כולם.

צעד 3: מנרמלים. לוקחים את ההופכי ומוצאים שורש מסדר N .

1.6.7 דוגמה מספרית: "החתול ישב"

נחשב את המורכבות של המשפט הקצר: "החתול ישב" (2 מילים).

צעד 1: מוצאים הסתברויות
המודל נותן את התוצאות הבאות:

- המילה "החתול" (בתחילת המשפט): $P = 0.01$

- המילה "ישב" (אחרי "החתול"): $P = 0.1$

למה 0.01? כי יש הרבה מילים שאפשר להתחיל איתן משפט. למה 0.1? כי חתול יכול גם ללקת, לרוץ, לאכול. לשבת זו אפשרות אחת מתוך רבות.

צעד 2: מכפילים

$$P(\text{ המשפט}) = 0.01 \times 0.1 = 0.001$$

צעד 3: מחשבים מורכבות

$$\begin{aligned} PP &= \sqrt[2]{\frac{1}{0.001}} \\ &= \sqrt{1000} \\ &= 31.6 \end{aligned}$$

מה המספר אומר?

המורכבות היא 31.6. המשמעות: המודל יהיה מבולבל כאילו היה עליו לבחור מתוך כ-32 מילים אפשריות בכל שלב.

1.6.8 השוואה: משפט הגיוני מול משפט מוזר

בואו נראה למה מורכבות נמוכה היא דבר טוב.

משפט א' (הגיוני): "ירץ גשם"

- הסתברות ל"ירץ": 0.05

- הסתברות ל"גשם" (אחרי "ירד") : 0.8 — מאוד צפוי!

$$P(\text{המשפט}) = 0.05 \times 0.8 = 0.04$$

$$PP = \sqrt{\frac{1}{0.04}} = \sqrt{25} = 5$$

משפט ב' (מוזר): "ירד פיל"

- הסתברות ל"ירד": 0.05

- הסתברות ל"פיל" (אחרי "ירד") : 0.0001 — מאוד מפתיע!

$$P(\text{המשפט}) = 0.05 \times 0.0001 = 0.000005$$

$$PP = \sqrt{\frac{1}{0.000005}} = \sqrt{200000} = 447$$

▲ ההבדל דרמטי

"ירד גשם": מרכיבות 5. המודל לא מופתע.
"ירד פיל": מרכיבות 447. המודל בהלם.

1.6.9 הקשר למשוואה 1

חזרו למשוואה 1. שימו לב לרכיב האחרון:

$$\dots + \gamma \cdot \text{Perplexity}(x)$$

הרכיב זהה מודד את **המורכבות של הפרומפט עצמו**. לא של התשובה. של הפרומפט x שאותם כתבים. מה קורה אם תכתבו פרומפט עם שגיאות כתיב? עם מיללים שלא קשורות? עם ניסוחים מסורבלים? המרכיבות תהיה גבוהה. המודל יתקשה "להבין" מה אתם רוצים.

הכלל הפשט

פרומפט טוב = מרכיבות נמוכה = שפה ברורה, תקנית וצפוייה.

פרומפט גרוע = מרכיבות גבוהה = שפה מבולבלת, משובשת או מוזרה.

המשוואה אומרת: כתבו פרומפטים פשוטים וברורים. אל תנסו להרשים את המודל עם שפה מליצית. הוא לא מתרשם. הוא רק מתבלבל.

משואה 2.1: מבחן צוואר הבקבוק המידעי

$$(4) \quad I(x; y) \leq \min\{H(x), H(y)\} - \epsilon \cdot \text{len}(x)$$

המשואה מבוססת על תורת המידע [4]. היא קובעת גבול עליון למידע שיכול לעבור מהפרומפט לתשובה.

- $I(x; y)$ = המידע הדדי בין הפרומפט לתשובה

- $H(x)$ = האנתרופיה של הפרומפט

- $H(y)$ = האנתרופיה של התשובה

- ϵ = קבוע דעיכה עבור פרומפטים ארוכים

אם $I(x; y)$ נמוך מדי, הפרומפט חסר מידע. אם גבוהה מדי, הפרומפט רועש.

משואה 3.1: השפעת המיקום

$$(5) \quad P_{correct} = \sigma(\theta_0 + \theta_1 \cdot \text{position}_{instruction} + \theta_2 \cdot \text{position}_{example})$$

המשואה מבוססת על מחקר [5] Lost in the Middle. היא מראה שהמיקום של המידע בפרומפט משפיע על הדיוק.

- σ = פונקציית סיגמוואיד (ממירה לסיכוי בין 0 ל-1)

- θ_0 = הטיה בסיסית

- θ_1 = משקל למיקום ההוראה

- θ_2 = משקל למיקום הדוגמאות

מחקרים מראים: $\theta_2 > \theta_1$ עבור מושימות חשיבה. ההוראה צריכה להיות בהתחלה. עבור מושימות יצירה, המגמה הפוכה: הדוגמאות חשובות יותר.

1.7 השוואת מטרות פרומפטים: Prompt Objectives Comparison

טבלה 1 מציגה סוגים שונים. לכל שימוש יש מאפיינים אחרים. המספרים מבוססים על מחקרים מ-2024 [1].

טבלה 2 משווה בין מודלים שונים. המספרים עדכניים ל-2025 ובסיסים על מחקרי שוק [6].

טבלה 1: השוואת מטרות פרומפטים לפי מדדי מידע

דוגמה לשימוש	סְפִי דַיּוֹק	Token Budget	Entropy	מטרת פרומפט
סיווג סנטימנט	0.95	50–100	< 2 bits	סיווג
כתיבת תקציר	0.85	200–500	5–7 bits	יצירה
פתרון מתמטי	0.90	400–800	< 3 bits	חשיבה
כתיבת פונקציות	0.92	300–600	< 4 bits	קוד

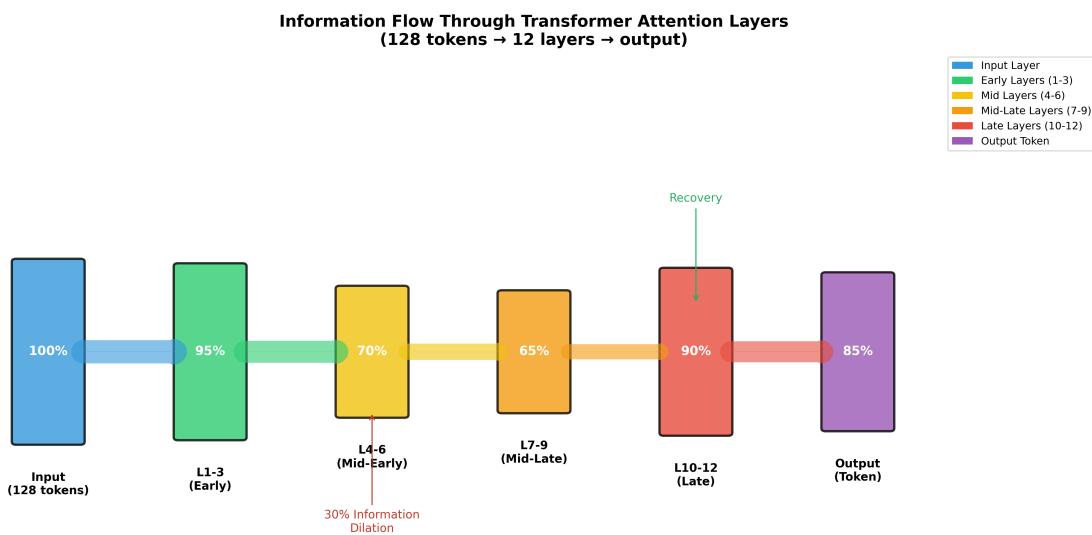
טבלה 2: מדדי אופטימליות פרומפט לדגמים שונים (2025)

טמפרטורה מומלצת	עלות	Attention Eff.	אורך אופטימי	Context	Model
0.6–0.8	\$3/M	0.87	800–1200	200K	Claude 3.5
0.5–0.7	\$5/M	0.91	600–900	128K	GPT-4o
0.4–0.6	\$2.5/M	0.94	1500–3000	1M	Gemini 2.5
0.6–0.9	\$0.8/M	0.85	500–800	128K	Llama 3.1

1.8 ויזואלייזציה של זרימת מידע: Information Flow Visualization

איך המידע זורם דרך המודל? הטרנספורמר [7] עובד בשכבות. כל שכבה מעבדת את המידע. חלק מהמידע נשמר. חלק אובד.

איור 1 מציג את האריםה. 128 טוקנים נכנסים. הם עוברים 12 שכבות. בסוף י יצא טוקן אחד. שימושו לב: בשכבה 6 יש ירידה של 30 אחוז בCAF של המידע. בשכבה 10 יש התאוששות חלקית.



איור 1: זרימת מידע דרך שכבות Attention בטרנספורמר

איור 2 מציג משטח תלת-ממדי. ציריו ה-X ו-Y הם אורך הפרומפט והטמפרטורה. ציר ה-Z הוא הדיק. אפשר לראות בבירור: ישאזור אופטימלי. מחוץ לאזור זה, הביצועים יורדים.

1.9 דוגמאות עשה ואל תעשה: Do and Don't Examples

הנה דוגמאות מעשיות. למדנו מיהן איך לכתוב פרומפטים טובים.

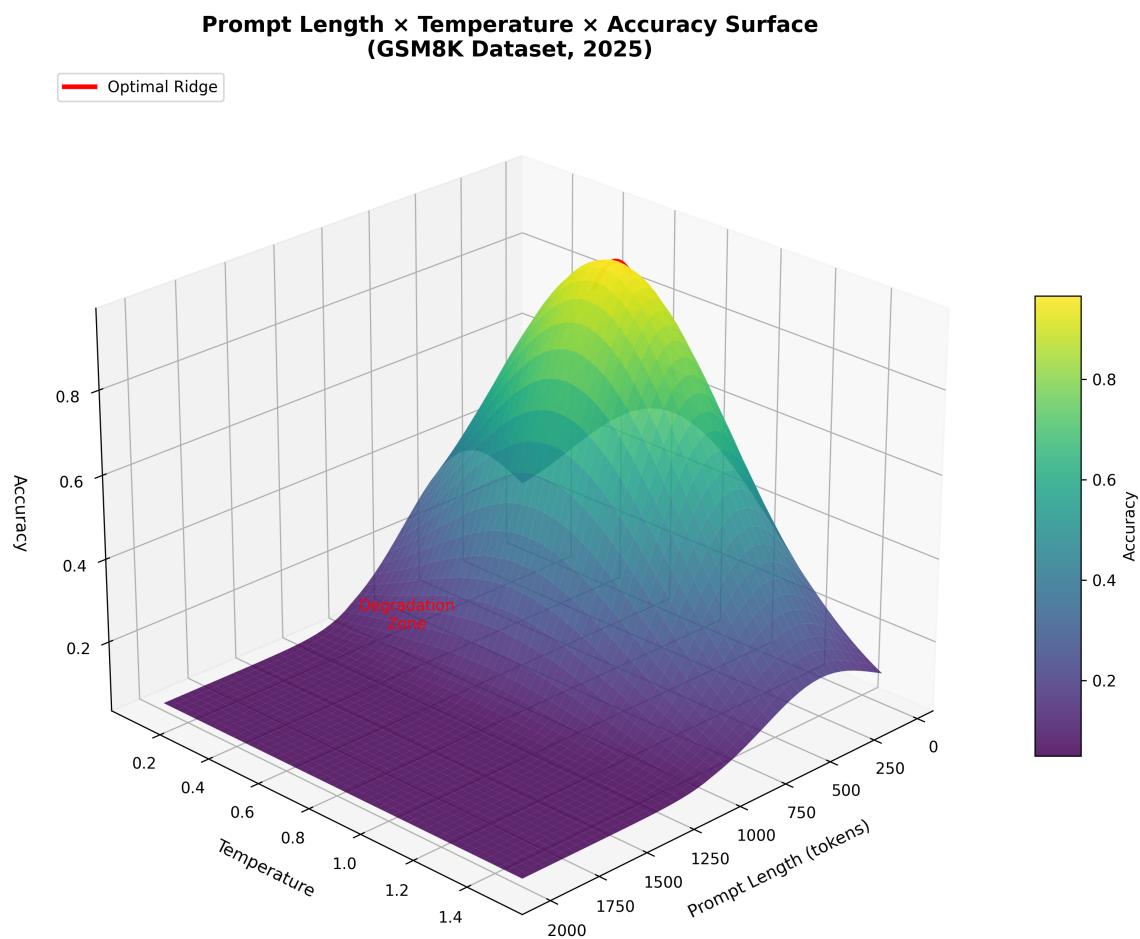
✓ עשה:

"כתב פונקציית פיתון שמחשבת את הממוצע המשוקל של רשימה. השתמש ב-type ".examples, וכתב docstring עם hints

▲ אל תעשה:

"תנו לי קוד לחישוב ממוצע."

הפרומפט הטוב ספציפי. הוא מגדר את השפה. הוא דורש תיעוד. הוא נותן הקשר. הפרומפט הגרוע מעורפל. המודל צריך לנחש מה אנחנו רוצחים.



איור 2: משטח דיק כפונקציה של אורך פרומפט וטמפרטורה (מערך נתונים GSM8K)

✓ עשה:

"הסביר את אלגוריתם Dijkstra. תחילה, הגדר את המשתנים: .graph, source, distances".distances של מערך ה-state לאחר כל שלב, הצג את ה-state

▲ אל תעשה:

"מה זה ?Dijkstra"

הפרומפט הטוב משתמש בשרשרת חסיבה [8]. הוא מבקש הסבר צעד אחר צעד. הוא מגידר את המבנה מראש.

✓ עשה:

"תרגם את הטקסט לאנגלית. שמור על רמת שפה פורמלית וטון מקצועני. הטקסט: 'הדווח הכספי מצבע על גידול של 15 אחוז ברווחים'."

▲ אל תעשה:

"תרגום: 'הדווח הכספי מצבע על גידול של 15 אחוז ברווחים'."

הפרומפט הטוב מגידר סגנון. הוא מציין את הרמה הרצויה. הוא נותן הקשר.

✓ עשה:

"השתמש בשרשרת חסיבה: קודם, חשב את השורש הראשון. אחר כך, הוסף אותו לממוצע. הסבר כל צעד."

▲ אל תעשה:

"פתרונות: $\sqrt{16} + \frac{1}{2}$ "

1.10 סיכום הפרק: Chapter Summary

בפרק זה למדנו את היסודות התיאורתיים של הנדסת פרומפטים. גילינו את הפרדוקס של העדרויות. פרומפט צריך להיות מאוזן. לא ספציפי מדי. לא כללי מדי. למדנו את משפט ויאו. תקשורת נכשלת בדרך כלל. لكن צריך לבדוק כל פרומפט. הבנו את מרחב ההסתברות. אורך הפרומפט משפיע על הדיקוק. יש נקודה אופטימלית. הצגנו שלוש משוואות מתמטיות:

- משואה 1: פונקציית האובדן לפромפטים

- משואה 4: צוואר הבקבוק המידי

- משואה 5: השפעת המיקום

בפרק הבא נעבור מתיאוריה לפרקтика. נלמד מודולוגיה מערכתית לתכנון פרומפטים. נראה איך לפרק פרומפטים למרכיבים אוטומטיים. בניית מערכת הערכה שיטתית.

2 מתודולוגיה מערכית לתוכנו פרומפטים

במאה העשרים, המדענים גילו סוד פשוט. כדי להבין מערכת מורכבת, צריך לפרק אותה לחלקים קטנים. האותם הפך למפתח להבנת החומר. הגן הפך למפתח להבנת החיים. ביום, אנו מגלים שהוא עיקרונו פועל גם בתקשורת עם מערכות AI.

פרומפט טוב אינו תוצאה של השראה פתאומית. הוא תוצאה של שיטה. פרק זה מציג את השיטה זו.

מטרות הלמידה

בסיום פרק זה, הקורא יוכל:

- לפתח framework מדיד לכנתיבת פרומפטים
- ליישם atomic prompting ו prompt decomposition
- לבנות prompt evaluation suite
- להשתמש בכלל שלושת הפרומפטים לאופטימיזציה

2.1 עקרון הפרומפט האוטומי Atomic Prompting Principle

עקרון הפרומפט האוטומי

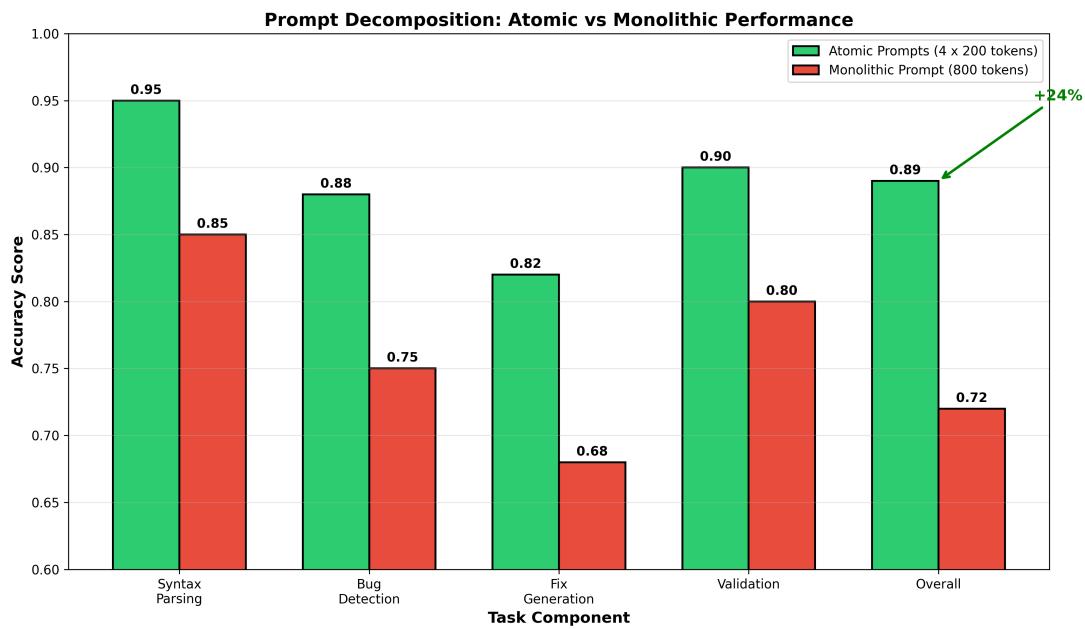
כל פרומפט חייב לבדוק היפותזה אחת בלבד. פרומפטים מורכבים הם הרכבה של אוטומים דרך אופרטורי composition.

מהו אוטום? בפיזיקה, אוטום הוא החלקיק הקטן ביותר שעדיין שומר על תכונות היסוד. בהנדסת פרומפטים, פרומפט אוטומי הוא ההוראה הקטנה ביותר שעדיין מבצעת משימה אחת מוגדרת [9].

בואו נחשב על משימה מורכבת: "ניתוח קוד". משימה זו כוללת למעשה ארבע משימות נפרדות:

1. ניתוח תחביר (Syntax Parsing)
2. זיהוי באגים (Bug Detection)
3. ייצור תיקון (Fix Generation)
4. אימות התיקון (Validation)

פרומפט מונוליטי מנסה לעשות הכל במת אחת. פרומפט אוטומי מפרק את המשימה לארבעה חלקים. מחקרים מראים שהגישה האוטומית משיגה דיקון גבוה יותר [1]. איור 3 מציג את ההבדל.



איור 3: השוואה בין פרומפטים אטומיים לפרומפט מונוליתי. הגישה האטומית משיגה דיקوك כולל של 89% לעומת 72% בגישה המונוליתית.

למה זה עובד? התשובה טמונה באופן שבו מודלי שפה מעבדים מידע. כאשר פרומפט מכיל שימושות רבות, המודל מחלק את תשומת הלב ביניהן. כאשר פרומפט מכיל משימה אחת, כל תשומת הלב מתמקדת בה [5].

טבלה 3: פירוק פרומפט לשימוש ניתוח קוד

כלי נדרש	דיקוק צפוי	תקציר	סוג פרומפט	רכיב
AST parser	0.95	100	Zero-shot	ניתוח תחביר
Static analyzer	0.88	200	Few-shot	זיהוי באגים
Code executor	0.82	300	Chain-of-Thought	יצירת תיקון
Test runner	0.90	150	Self-consistency	אימונות

2.1.1 פירוש הטבלה: ארבעה שלבים בתהליכי העבודה

טבלה 3 מציגה תהליך עבודה שלם לפיתוח תוכנה בעזרת AI. כל שורה מייצגת שלב אחר. כל שלב דורש סוג פרומפט שונה וכלי שונה. זהו אמונות הפירוק לאוטומים בפועל.

שלב ראשון: ניתוח תחביר המשימה הראשונה היא הפשטה ביותר: לבדוק אם הקוד כתוב בצורה תקינה. האם חסר סוגר? האם יש נקודה-פסיק במקום הנכון? שאלות אלה הן

בニアריות. התשובה היא "כן" או "לא".

סוג הפרומפט: Zero-shot – פניה למודל ללא דוגמאות מקדימות. מדוע? בדיקת תחביר היא משימה בסיסית. המודל אינו זוקק לדוגמאות כדי לדעת אם חסר סוגר. הפרומפט יהיה פשוט: "האם הקוד הבא תקין תחבירית?"

הכלי הנדרש: AST parser – מנתח עץ תחביר מופשט. כלי זה אינו מבוסס AI. הוא לוקח את הקוד ומנסה לבנות ממנו "עץ" היררכי. אם הבנייה נכשלה, יש שנייה תחביר. זהו כלי דטרמיניסטי. הוא מדויק ב-100%.

שלב שני: זיהוי באגים המשימה השניה מורכבת יותר. אנחנו מחפשים שגיאות לוגיות. הקוד עובר את בדיקת התחביר, אבל האם הוא עושה מה שהוא צריך לעשות?

סוג הפרומפט: Few-shot – פרומפט כולל 2–3 דוגמאות של קלט ופלט. למשל: "הנה קוד עם באג → הנה הבא שנמצא". הדוגמאות עוזרות למודל להבין איזה סוג של באגים אנחנו מחפשים. האם אבטחה? ביצועים? לוגיקה?

הכלי הנדרש: Static analyzer – מנתח סטטי. כלי שסורק את הקוד מבליל להריצ' אותו. הוא דומה לבודק איות, אבל לקוד. כלים מוכרים הם ESLint, Pylint, SonarQube, ו-1. מזהים תבניות בעייתיות ידועות מראש.

שלב שלישי: יצירת תיקון המשימה השלישית היא הקשה ביותר. המודל צריך לכתוב קוד מתוקן. זו כבר לא בדיקה. זו יצירה.

סוג הפרומפט: Chain-of-Thought – שרשרת חשיבה. אנחנו מנחים את המודל "לחשוב בקורס". לפרק את הבעיה לשלבים לפני שהוא כותב את הקוד. אם המודל ינסה ישר לפלוט קוד, הוא עלול לטעות. אבל אם הוא קודם יכתוב: "1. הבא נושא מלולאה אינסופית. 2. צריך להוסיף תנאי עצירה. 3. הנה הקוד המתוקן" – הדיקוק עולה ממשמעותית [8].

הכלי הנדרש: Code executor – מרייך קוד. סביבת Sandbox שבה ניתן להריץ את הקוד שהמודל כתב. זהו המבחן המעשי הראשון. האם הקוד בכלל עובד? האם הוא קורס?

שלב רביעי: אימות המשימה האחרונה היא קריטית. תיקון נכתב. הקוד רץ. אבל האם הוא באמת פתר את הבעיה? האם הוא לא יצר בעיות חדשות?

סוג הפרומפט: Self-consistency – עקבות עצמית. טכניקה שבה מבקשים מהמודל לייצר את הפתרון מספר פעמים. למשל, 5 פעמים. אז בוחרים את התשובה הנpostaה ביותר. אם המודל הציע את אותו פתרון 4 מתוך 5 פעמים, הסבירות שהוא נכון גבוהה מאוד [10].

הכלי הנדרש: Test runner – מרייך בדיקות. כל שMRIICH סט של בדיקות אוטומטיות Unit Tests) שהוגדרו מראש. הוא נותן את החותמת הסופית: " עבר" או "נכשל". אם הקוד עבר את כל הבדיקות, תיקון מאושר.

זה תהליך מלא. ארבעה שלבים. ארבעה סוגי פרומפט. ארבעה כלים. כל אחד מותאם למשימה הספציפית שלו. זו המהות של גישת הפרומפט האוטומי.

2.2 השוואת פרומפטים: Prompt Diffing

Prompt Diffing

השווות שני פרומפטים על אותו input כדי לאוזות מרכיבים קרייטיים. כל שינוי מעלה 5% ב-**Root Cause Analysis accuracy**.

לפניהם רפואיים מחליטים איזו תרופה עיליה יותר, הם עורכים ניסוי מבוקר. קבוצה אחת מקבלת את התרופה. קבוצה אחרת מקבלת פלאצבו. ההבדל בתוצאות מגלת את האפקט האמתי.

הנדסת פרומפטים דורשת אותה קפדות מדעית. כאשר משנים פרומפט, צריך למדוד את ההשפעה. זהה טכניקת **Prompt Diffing** [11].

התהליך פשוט:

1. קח פרומפט קיים (baseline)

2. שנה אלמנט אחד בלבד

3. הרץ את שני הפרומפטים על אותם נתונים

4. מדוד את ההבדל בדיק

אם ההבדל גדול מ-5%, יש לבצע ניתוח שורש הבעיה. מה בשינוי גרם לשיפור או להרעה? התשובה מלמדת אותנו מה באמת חשוב בפרומפט.

טבלה 4 מציגה דוגמה לשיפור איטרטיבי. כל גרסה משפרת את הקודמת.

טבלה 4: מדדי שיפור איטרציה-אחר-איטרציה

איטרציה	גרסה	דיוק	היזות	השהייה	שינוי
1	v0.1	0.72	0.15	1200ms	-
2	v0.2	0.79	0.12	1350ms	+9.7%
3	v0.3	0.85	0.08	1800ms	+7.6%
4	v0.4	0.89	0.05	2100ms	+4.7%

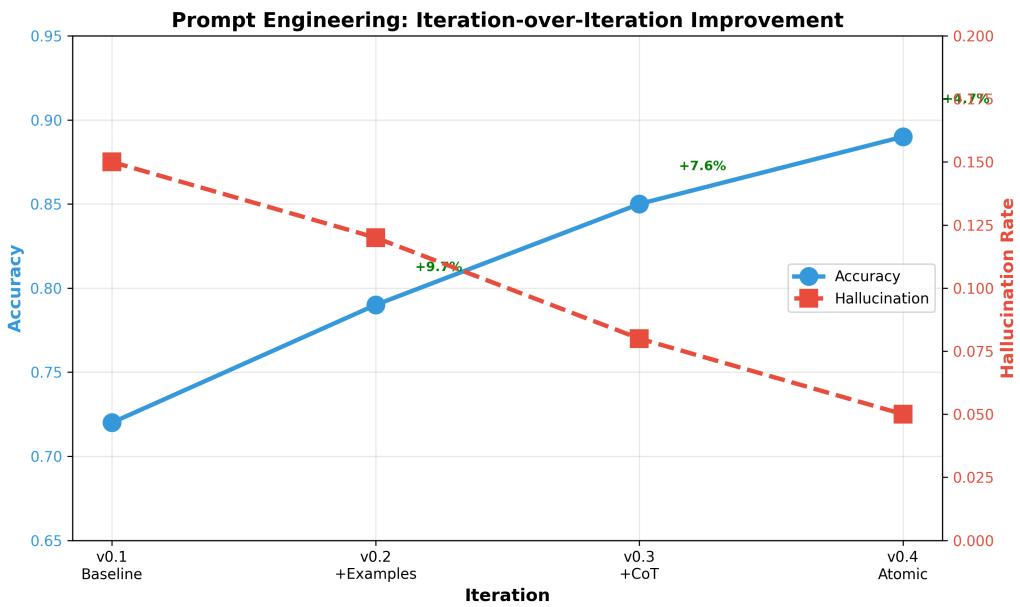
כפי שאייר 4 מראה, הדיוק עולה מ-72% ל-89%. במקביל, שיעור ההיזות יורדת מ-15% ל-5%. זהו שיפור משמעותי שנבנה צעד אחר צעד.

2.3 כלל שלושת הפרומפטים: The 3-Prompt Rule

כלל שלושת הפרומפטים

לכל שימושה, כתוב שלושה פרומפטים:

- tokens 50 – Minimal -



איור 4: שיפור איטרטיבי: דיקוק עולה והזיות יורדות לאורך ארבע איטרציות של שיפור פרוומפט.

tokens 200 – Balanced -

tokens 500 – Comprehensive -

בדוק أيיה מודל מגיב הכி טוב לכל אחד.

הסיפור של זהבה ושלושת הדובים מלמד אותנו עיקרון חשוב. לא קטן מדי, לא גדול מדי – בדיקות מתאימים. עיקרון זה תקף גם לפרוומפטים. מחוקרים מ-2025 מראים תופעה מפתיעה: יותר מידי לא תמיד אומר תוצאות טובות יותר [12]. לעיתים פרוומפט קצר עובד טוב יותר מפרוומפט ארוך. הכל תלוי במשימה ובמודל.

לכן, הכלל הוא פשוט: כתוב שלוש גרסאות. בדוק את כלן. בחר את הטובה ביותר. איור 5 מציג את מחזור החיים המלא של הנדסת פרוומפטים. שלב ההערכתה הוא קריטי: רק פרוומפט שעובר את סף הדיקוק מגיע לייצור.

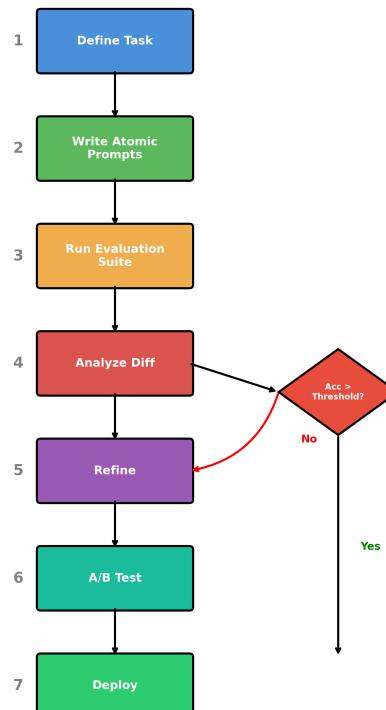
2.4 נוסחאות מתמטיות: Mathematical Formulas

כדי למדוד את יעילות הפירוק לפרוומפטים אוטומטיים, אנו משתמשים בציון פירוק פרוומפט (Prompt Decomposition Score).

משוואה 2.1: ציון פירוק פרוומפט (PDS)

$$(6) \quad PDS = \frac{\sum_{i=1}^n \text{Acc}(p_i) \cdot \log(\text{len}(p_i))}{\log(\text{len}(p_{full}))}$$

Prompt Engineering Lifecycle



איור 5: מחזור החיים של הנדסת פרומפטים: מהגדרת משימה ועד פרישה ביצוע.

כאשר p_i = פרומפט אוטומי, p_{full} = פרומפט מונוליטי. ציון $PDS > 1.15$ מצביע על יתרון decomposition.

הנוסחה משקלה את הדיק של כל פרומפט אוטומי לפי אורכו. פרומפט קצר עם דיק גבוה מקבל ציון גובה יותר. אם הציון הכלול עולה על 1.15, הפירוק לאוטומים משתלם. כדי לדעת אם שיפור בדיק הוא אמיתי ולא מקרי, אנו משתמשים בנוסחת רוח הסמך.

משוואת 2.2: רוח בר-סמק לשיפור פרומפט

$$(7) \quad \Delta_{\text{significant}} = z \cdot \sqrt{\frac{p_1(1-p_1)}{n_1} + \frac{p_2(1-p_2)}{n_2}}$$

עבור $z = 1.96$ (רמת סמך 95%), שיפור ב-accuracy נחשב מובהק רק אם $\Delta > \Delta_{\text{significant}}$.

לדוגמא: אם בדקנו 1000 דוגמאות, שיפור של 2% בדיק הוא מובהק סטטיסטי. שיפור של 0.5% עשוי להיות מקרי.

2.5 דוגמאות עשה ואל תעשה: Do and Don't Examples

הבדל בין פרומפט טוב לפרומפט גרוע הוא לעיתים קטן אך משמעותי. הנה דוגמאות מעשיות.

✓ עשה: Atomic Decomposition

"משימה 1: חלץ את כל שמות המשתנים מהקוד הבא. רשום אותם כ-bullet points.
קוד: ..."

▲ אל תעשה: Monolithic

"תנתח את הקוד, מצא באגים, תקן אותם, וכתוב דוח."

הפרומפט הראשון מבקש משימה אחת ברורה. הפרומפט השני מבקש ארבע משימות בבת אחת.

✓ עשה: Explicit Steps

"שלב 1: קרא את השאלה. שלב 2: חשב את ה-derivative. שלב 3: הכנס את $x = 2$.
שלב 4: בדוק את התשובה."

▲ אל תעשה: Implicit

"פתרו את השאלה."

הפרומפט הראשון מפרט את הצעדים. הפרומפט השני מניח שהמודל ידע מה לעשות. לעיתים הוא יודע. לעיתים לא.

✓ עשה: Validation Loop

"כתב פונקציה. אחרי הכתיבה, הרץ אותה עם $input=5$. אם התוצאה $\neq 15$, תקן את הלוגיקה."

▲ אל תעשה: No Validation

"כתב פונקציה."

הפרומפט הראשון כולל מנגנון בדיקה עצמית. הפרומפט השני לא. מחקרים מראים ש-self-validation מפחית טעויות ב-30%.[13]

✓ עשה: Diff Analysis

"השווה בין שני הפרומפטים הבאים. הצג הבדלים ב-latency, accuracy, ו-hallucination.".rate

▲ אל תעשה: No Comparison

"בודק איזה פרומפט טוב יותר."

הפרומפט הראשון מגדיר קритריונים ברורים. הפרומפט השני מותיר את החלטה למודל.

2.6 סיכום הפרק: Chapter Summary

פרק זה הציג שיטה מערכתית לתכנון פרומפטים. שלושה עקרונות מרכזיים:

1. **עקרון האטום:** פרק משלימות מורכבות לפרומפטים פשוטים. כל פרומפט עושה דבר אחד. הרכיב אותם יחד לפתרון שלם.

2. **השווואה מדעית:** השווה פרומפטים באופן מבויק. שנה משתנה אחד בכל פעם. מודוד את ההשפעה.

3. **כל השלישי:** כתוב שלוש גרסאות. בדוק את כלן. בחר את המתאימה ביותר למשימה ולמודל.

בפרק הבא נלמד על תבניות פרומפט מתקדמיות: Tree-of-, ReAct, Chain-of-Thought. תבניות אלה מרחיבות את הכלים שלנו לייצרת פרומפטים עילים יותר. Thoughts

3 תבניות פרומפט מתקדמות וארכיטקטורות

מטרות הלמידה

בסיום פרק זה, הקורא יוכל:

- לנתח לעומק את Tree-of-Thoughts, ReAct, Chain-of-Thought
- לישם few-shot learning אופטימלי
- לבנות מערכות role-based prompting
- להבין את היתרונות והחסרונות של כל תבנית

3.1 מבוא: תפקיד המתכנת בעידן ההסקה

מודל שפה גדול הוא יוצר מוזר. הוא יודע הכל, אבל לא יודע לחשב. נסbir. כשהאתה שואל אותו שאלה, הוא קופץ לתשובה. הוא לא עוצר לחשב. הוא לא שואל את עצמו: "האם אני בטוח?". הוא פשוט עונה. לעיתים הוא צודק. לעיתים הוא טועה. ולפעמים הוא ממציא דברים שלא היו ולא נבראו. החוקרים קוראים לזה "הזיות" (Hallucinations). אבל הנה הבעיה האמיתית: אתה לא יכול לתקן את זה. לא באמת. אתה לא יכול לפתח את המודל ולשנות אותו. אתה לא יכול לאמן אותו מחדש. זה עולה מיליון. זה לוקח חודשים. אתה עובד ברמת ההסקה (Inference) בלבד. המודל הוא קופסה שחורה. אתה יכול רק לדבר אליו. וכך נכנס המתכנת לתמונה.

3.1.1 הפיגום החיצוני

אם אתה לא יכול לשנות את המוח – שנה את הסביבה. חשוב עלILD עם הפרעת קשב. הוא לא טיפש. הוא פשוט קופץ מחשבה למחשבה. מה עושים? נתונים לו מבנה. רשימת משימות. שלב אחרי שלב. אותו דבר עם מודלי שפה. הטכניקות שנלמד בפרק זה – Tree-of-Thoughts, ReAct, Chain-of-Thought – עושות בדיקות זה. הן בונות פיגום חיצוני (External Scaffolding) סביב המודל. הפיגום מכתיב לו איך לחשב. הוא לא משנה את המודל. הוא משנה את השיחה [8].

3.1.2 שני מישורים של בקרה

יש שתי דרכי לבנות פיגום.
הדרך הראשונה: הנחיה מובנית (Structured Prompting). אתה כותב בפרומפט: "חשוב צעד אחר צעד". "הסביר את ההיגיון שלך". "אל תקופץ למסקנה". המודל מקשיב. הוא מאט. הוא מראה את העבודה. זה העובדה. זה עובד. זה פשוט. Chain-of-Thought

הדרך השנייה: לולאות בקרה חיצונית. לעיתים הנחיה לא מספיקה. לעיתים צריך קוד. ReAct, המודל צריך לגשת לכלים חיצוניים. מנוע חיפוש. מחשבון. מאגר נתונים. מי מנהל את זה? לא המודל. אתה. ב-Tree-of-Thoughts, המודל צריך לחזור מספר נתיבים במקביל. מי מחליט איפה נתיב גוזם? לא המודל. אתה. אתה כותב קוד שמריץ את המודל. שולח לו שאלות. מקבל תשובות. מעיריך אותן. מחליט מה לעשות להלה. הקוד שלך הוא הפיגום. הוא מנהל את המצב. הוא מתאים את התהיליך [14].

3.1.3 המסקנה החשובה

המודל לא יוזם את הטכניקות האלה בעצמו. זה שהוא לא יודע שהוא צריך. הוא אומן על טקסטים. הוא למד לחזות את המילה הבאה. אף אחד לא לימד אותו לעצור ולחשוב.
זה הפקיד שלו.

אתה המתכוна. אתה מגדיר את הפונקציה של הפרומפט. אתה קובע את הגבולות. אתה בונה את הפיגום שהופך מודל שkopf למסקנות – למודל חשוב.
בפרק זה נלמד שלוש תבניות מרכזיות:

1. Chain-of-Thought – חשיבה צעד אחר צעד, באמצעות הנחיה מובנית

2. ReAct – שילוב חשיבה עם פעולה, באמצעות לולאות בקרה חיצונית

3. Tree-of-Thoughts – חקירת מספר נתיבים, באמצעות קוד שמנהל את העץ

3.2 :שרשרת מחשבה

3.2.1 הגדרה פורמלית

הגדרה:]

טכנית הנדסת פרומפטים המאפשרת למודלי שפה גדולים [Chain-of-Thought (CoT) (LLMs) להתמודד עם משימות חשיבה מורכבות. השיטה יוצרת רצף ליניארי של שלבי בניינים לוגיים ("מחשבות"). השלבים מושרים בין הקלט לבין הפלט הסופי, במקומות לחזות את התשובה באופן ישיר.

3.2.2 הסבר נרטיבי

איך אנחנו פותרים בעיה קשה? אנחנו לא קופצים לתשובה. אנחנו חושבים בשלבים. קחILD שלומד חשבון. הוא לא כותב רק "התשובה היא 42". הוא כותב: "קודם נחבר את האחדות. אחר כך נעביר לעשרות". השלבים האלה הם לא סתם נוחות. הם הדרך שבה המוח עובד.

מודלי שפה למדו מטקסטים של בני אדם. הם רואו מיליארדי דוגמאות של תשובות. אבל רוב התשובות האלה הן "קפיצה" ישירה למסקנה. בלי הסבר. בלי שלבים. כשבקשים מהמודל "לחשוב בקול רם" – הכל משתנה [8].

3.2.3 דוגמה פשוטה

✓ **עשה:** עם Chain-of-Thought שאלתך: "לدني 5 תפוחים. הוא זרק 2 ומצא 3. כמה יש לו?"

תשובה המודל:

בהתחלת היו 5 תפוחים.

הוא זרק 2, אז נשארו 3.

הוא מצא 3, אז 3 ועוד 3 זה 6.

התשובה הסופית: 6 תפוחים.

▲ **אל עשה:** בלי Chain-of-Thought

שאלתך: "לدني 5 תפוחים. הוא זרק 2 ומצא 3. כמה יש לו?"

תשובה המודל: "6".

(המודל נותן תשובה נכונה, אבל אין דרך לבדוק את ההיגיון)

3.2.4 גרסה מתקדמת: Chain-of-Thought++

Chain-of-Thought++

עם CoT קיימים 3 נתיבי חשיבה, הצבעת רוב. ב-2025 הוכח self-consistency voting ש-5 נתיבים עם temperatue-Sharing מוסיפים 12% ל-accuracy.

3.2.5 מהות השיטה המתקדמת

בני אדם לא קופצים למסקנות. הם חושבים בשלבים. כשהאנחנו פותרים בעיה מתמטית, אנחנו כתבים את השלבים. כשהאנחנו מנתחים טקסט, אנחנו מפרקים אותו לחלקים. Chain-of-Thought מלמד מודלים לעשות את אותו הדבר [8].

השיטה פשוטה. במקום לבקש תשובה ישירה, מבקשים מהמודל להסביר את החשיבה. במקרה "מה התשובה?", שואלים "חשוב צעד אחר צעדים, ואז תן תשובה". ההבדל זהה משנה הכל.

Weiyi ועמיתיו ב-Google הראו תוצאות מדדיימות [8]. על מבחן GSM8K של בעיות מתמטיות, הדיקוק עלה מ-17.9% ל-58.1%. זה שיפור של יותר מפי שלושה.

3.2.6 הצבעת עקבות עצמית

אבל יש בעיה. לעיתים המודל טועה בשלב אחד. הטעות גוררת טעויות נוספות. הפתרון: ליצר מספר נתיבי חשיבה ולהציג על התשובה הנפוצה ביותר. הפתרון: Wang ו עמיתיו קראו לשיטה Self-Consistency [10]. הרעיון פשוט. מרכיבים את אותו פרומפט 5 פעמים עם temperature שונה. כל הרצה מייצרת נתיב חשיבה שונה. בסוף, בוחרים בתשובה שהופיעה הכי הרבה. המשוואה 8 מתארת את החישוב המתמטי:

משוואה 1.3: הצבעת CoT Self-Consistency ב- \hat{y}

$$(8) \quad \hat{y} = \arg \max_{y \in Y} \sum_{i=1}^N \mathbb{1}[y_i = y], \quad y_i \sim P(y|x, \tau_i)$$

במשוואה זו, N הוא מספר הנתיבים. τ_i הוא ה-temperature של כל נתיב. $\mathbb{1}[y_i = y]$ היא פונקציית אינדיקטור שסופרת כמה פעמים הופיעה כל תשובה.

3.3 ReAct: שילוב חשיבה ופעולה

3.3.1 הגדרה פורמלית

הגדרה:]

פרדיגמה המשלבת יכולות הסקה (Reasoning) עם ביצוע פעולות (Acting). השיטה מאפשרת למודל שפה ליצור עקבות חשיבה מילוליות ולבצע פעולות מול כלים חיצוניים. הכלים יכולים להיות מנויי חישוב, מחשבונים, או מאגרי מידע. התהליך הוא משולב, אינטראקטיבי ומהזורי – עד להשלמת המשימה.

3.3.2 הסבר נרטיבי

מודל שפה הוא כמו מוח באנצנטרה. הוא יודע הרבה, אבל הוא מנוטק מהעולם. חשוב על זה כך: אם תשאל אותו "מה מזג האוויר בפריז עכשו?" – אני לא יודעת. אני יכול לנחש, אבל אין לי גישה למידע עדכני. אני צריך לחפש. ReAct נותן למודל את יכולת זה [14].

השיטה עובדת בمعالג פשוט:

1. **מחשבה** (Thought): "מה אני צריך לדעת?"

2. **פעולה** (Action): שימוש בכלים חיצוניים להשגת המידע

3. תצפית (Observation): קריית התוצאה שוחרה

המיגל חוזר על עצמו. המודל חושב, פועל, צופה, וחוזר לחשוב. זה הופך אותו מ"מוח באנצננת" ל"סוכן פעל" בעולם.

3.3.3 דוגמה פשוטה

✓ **עשה:** ReAct בפועלה

שאלה: "האם יורד גשם בעיר הבירה של צרפת?"

מחשבה: אני צריך לדעת מה הבירה של צרפת, אז לבדוק מזג האוויר.

פעולה 1: search("capital of France")

תצפית: פריז.

פעולה 2: search("weather Paris now")

תצפית: שמי, 25 מעלות.

תשובה סופית: לא, בפריז קרונע שימושי.

3.3.4 גרסה מתקדמת: ReAct 2.0

ReAct 2.0

שינויי מ-*Reflect-Act-Observe* ל-*Think-Act-Observe*. שלב ה-*Reflect* מוסיף שכבה מטא-קוגניטיבית.

3.3.5 מעבר לפרזיגמה חדשה

חשיבה לבדה לא מספיקה. לפחות ציריך לפעול בעולם. ReAct משלב את שניהם: חשיבה (Reasoning) ופעולה (Acting) [14].

המודול חושב. הוא מחליט על פעולה. הוא מבצע אותה. הוא צופה בתוצאה. אז הוא חוזר לחשוב. הלולאה הזאת ממשיכה עד שהמשימה הושלמה.

Yao ועמייתיו-Princeton-Google פרסמו את השיטה ב-ICLR 2023 [14]. על HotpotQA, היא שיפרה את ההצלחה ב-34%.

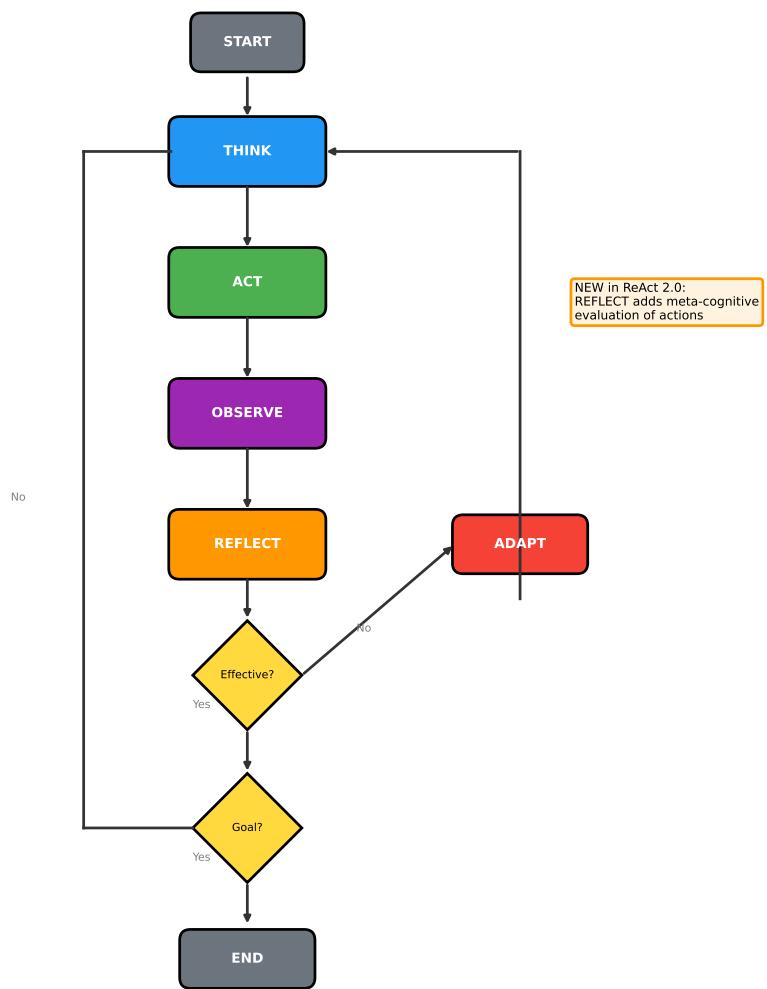
3.3.6 ארכיטקטורת הזרימה

ב-ReAct המקורי היו שלושה שלבים: חשב, פעל, צפה (Think-Act-Observe). ב-2.0 נוסף שלב רביעי: רפלקציה (Reflect).

השלב החדש שואל: "האם הפעולה הייתה אפקטיבית?". אם כן, ממשיכים. אם לא, ממשיכים אסטרטגייה (Adapt). זהה שכבה מטא-קוגניטיבית. המודול לומד לחשב על החשיבה שלו.

איור 6 מציג את הארכיטקטורה המלאה.

Figure 3.2: ReAct 2.0 Architecture
Think-Act-Observe-Reflect-Adapt Loop



איור 6: ארכיטקטורת ReAct 2.0 מעגל Act-Observe-Reflect-Adapt :ReAct 2.0

3.4 עץ מחשבות :Tree-of-Thoughts

3.4.1 הגדרה פורמלית

הגדרה:

Chain-of-Thoughts (ToT) מסגרת אלגוריתמית המכילה את גישת Tree-of-Thought. השיטה מאפשרת למודל לבחור מספר נתיבי חשיבה אפשריים במקביל ("ענפים"). המודל מבצע הערכתה עצמית (Self-Evaluation) לכל צומת בעץ. הוא מחליט האם להמשיך בתיב מסויים, לחזור אחורה (Backtrack), או לפצל את החשיבה לכיוונים חדשים.

3.4.2 הסבר נרטיבי

אם Chain-of-Thought הוא קו ישר, Tree-of-Thought הוא מבוק. חשוב על שחמטאי. הוא לא חושב רק על המהלך הבא. הוא חושב על עשרה מהלכים קדימה. יותר מזה – הוא בוחן כמה אפשרויות במקביל. "אם אני עשה כך, מה יקרה? ואם עשה אחרת?" Tree-of-Thought נותן למודל את יכולת זה [15]. המודל בונה עץ החלטות. בכל צומת יש מחשיבה – פתרון חלק. הוא נותן ציון לכל כיוון: "אם הכוון זהה מבטיח?" אם הוא מגיע למבי סתום, הוא ידוע לחזור אחורה ולנסות כיוון אחר.

3.4.3 דוגמה פשוטה

✓ **עשה:** Tree-of-Thought בפועל
משימה: כתוב משפט סיום למותחן.

שלב 1 – יצירת ענפים:

- המודל מציע 3 רעיונות:
(א) הרוצה הוא השוטר
(ב) הכל היה חלום
(ג) החיזירים פלשו

שלב 2 – הערכתה:

- (ב) קלשאתי → ציון נמוך
(ג) לא קשור לזרן → ציון נמוך
(א) מעניין ומשמעותי → ציון גבוה

שלב 3 – המשך:

המודל בוחר ב-(א) וממשיך לפתח רק אותו.

3.4.4 גרסה מתקדמת: Tree-of-Thought עם תקציב קבוע

Budget Forcing עם Tree-of-Thoughts

הגבלת מספר הצמתים ל- $B = 2^{\text{depth}}$ עם pruning דינמי.

3.4.5 חקירה מבנית

לפעמים יש יותר מדרך אחת לפתרו בעיה. Chain-of-Thought בוחר דרך אחת. Tree-of-Thoughts חוקר מספר דרכים במקביל [15]. הרעיון: לארגן את החשיבה עצ. כל צומת הוא "מחשבה" – פתרון חלק. המודל מעריך כל מחשבה ומחלית אם להמשיך או לגזום. התוצאות מרשים. על משחק Game of 24, GPT-4 עם CoT פתר 4% בלבד. עם Tree-of-Thoughts, ההצלחה עלה ל-74%. זו קפיצה של פי 18.

3.4.6 גיזום דינמי

ע"כ יכול לגדול ללא גבולות. זה בזבוז משאבים. לכן משתמשים בגיזום (pruning). הרעיון: לחתוכ ענפים עם ציון נמוך. להתמקד בענפים מבטיחים. משועואה 9 מגדירה את תנאי הגיזום:

משועואה 2.3: תנאי Pruning ב-Tree-of-Thoughts

$$(9) \quad \text{Prune}(n) = \begin{cases} \text{True} & \text{if } \frac{\sum_{i=1}^k \text{score}(c_i)}{k} < \lambda \cdot \text{score}(n) \\ \text{False} & \text{otherwise} \end{cases}$$

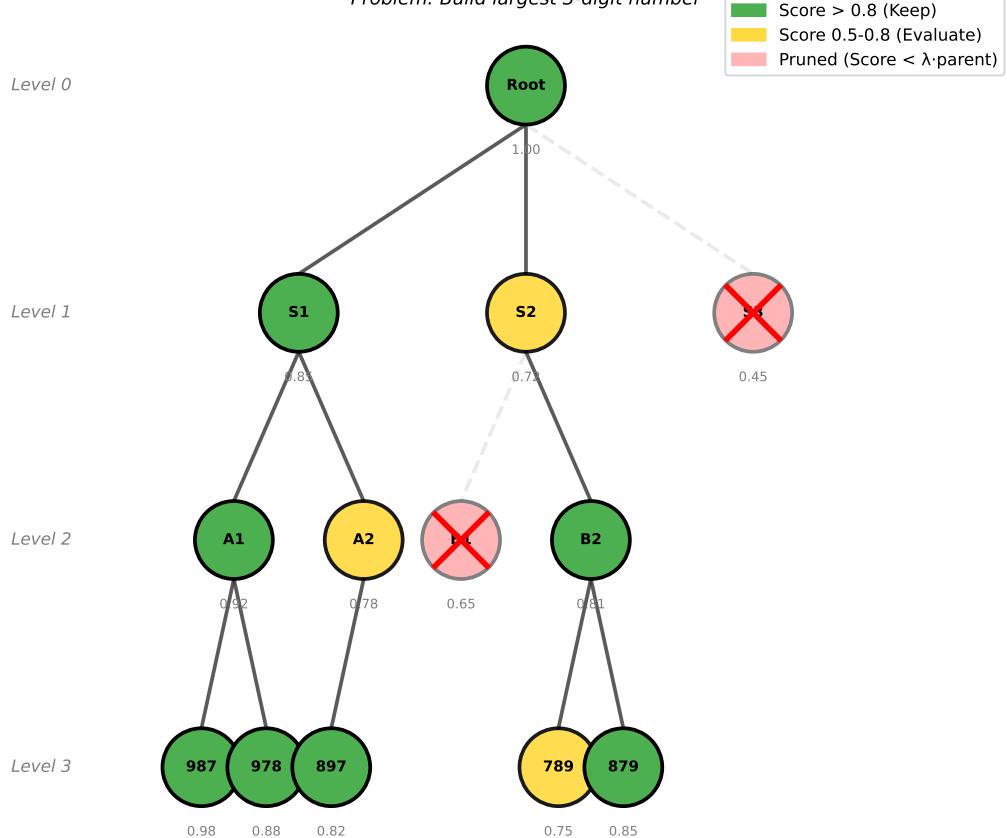
במשועואה זו, n הוא הצומת הנבדק. c_i הם הילדים שלו. $\lambda = 0.7$ הוא סף האיכות. אם ממוצע ציוני הילדים נמוך מ-70% מציון ההורה, גוזמים. איור 7 מציג דוגמה לעץ עם גיזום.

3.5 הנחיתת תפקידיים: Role-Based Prompting

האם למודל יש "אישיות"? לא באמת. אבל אפשר לתת לו תפקיד. "אתה פרופסור לכלכלה". "אתה מתכנת מומחה". "אתה מורה לילדים". התפקיד משפייע על התשובות. Kong ועמיתיו חקרו את ההשפעה [16]. על מבחן AQUA, הדיקוק עלה מ-53% ל-63%. אבל לא תמיד יש שיפור. Zheng ועמיתיו הראו שבחלק מהמקרים אין הבדל [17]. המשקנה: התפקיד עוזר כשהוא ספציפי ורלוונטי למשימה. "אתה מומחה" פחות טוב מאשר מומחה לכלכלה התנהגותית שמלמד סטודנטים". משועואה 01 מתארת את ההשפעה:

**Figure 3.1: Tree-of-Thoughts with Budget Forcing
($\lambda = 0.7$, $B = 2^{\text{depth}}$)**

Problem: Build largest 3-digit number



איור 7: עץ מחשבות עם גיזום: ענפים עם ציון נמוך נגזמים

משואה 3.3: השפעת Role-Based Prompting

$$(10) \quad \Delta_{\text{role}} = \mathbb{E}[\text{Acc}_{\text{with role}}] - \mathbb{E}[\text{Acc}_{\text{no role}}] = \beta \cdot \text{RoleSpecificity} + \epsilon$$

במשואה זו, β מודד את עוצמת ההשפעה. מחקרים מראים $\beta = 0.12$ למשימות חשיבה. למשימות יצירה, ההשפעה חלשה יותר: $\beta = 0.06$ [18].

3.6 למידה מדוגמאות: אופטימיזציה של Few-Shot

כמה דוגמאות צריך? זו שאלת חשובה. יותר מדי – בזבוז. פחות מדי – חוסר דיוק. OpenAI ועמיティו ב-Brown הראו את הכוח של למידה מדוגמאות [19]. 175 GPT-3 עם מיליארד פרמטרים למד מ-10-100 דוגמאות בלבד. לא היה צורך באימון נוספת. הדוגמאות הספיקו.

טבלה 5 מציגה את ההשפעה של מספר הדוגמאות:

טבלה 5: אופטימיזציה few-shot learning

השפעת סדר	שיעור דיוק	תקציב טוקנים	מספר דוגמאות
לא רלוונטי	+5%	+150	1
השפעה גבוהה	+11%	+450	3
השפעה בינונית	+14%	+750	5
רעש	+15%	+1200	8

המסקנה: 3 דוגמאות הן נקודת האיזון האופטימלית. יותר מכז – תשואה פוחתת.

3.7 השוואת מערכת בין תבניות

איזו תבנית לבחור? התשובה תלוי במשימה.

טבלה 6 משווה את התבניות השונות:

הכלל: התחל פשוט. Zero-shot לא עובד? נסה Few-shot. עדין לא מספיק? עבר ל-CoT. צריך לגשת למידע חיצוני? השתמש ב-ReAct. בעיה מורכבת עם מספר פתרונות? נסה ToT.

3.8 דוגמאות עשה ואל תעשה: Do and Don't Examples

✓ **עשה:** ReAct Structured

"שאלה: מה המطبع של צ'כיה?

טבלה 6: השוואת תבניות פרומפטים לפי סיבוכיות

מתאים למשימות	מכפיל זמן	שיפור דיוק	עומס טוקנים	תבנית
פשטות	1.0x	בסיס	1.0x	Zero-shot
דורשות דוגמאות	1.0x	+8-12%	1.3x	Few-shot
חשיבה, חשבון	1.5x	+15-20%	1.8x	CoT
כלים, חיפוש	2.0-3.0x	+18-25%	2.5x	ReAct
יצירתיות	4.0-6.0x	+22-30%	4.0x	ToT

מחשבה: צ'כיה היא באירופה, אני צריך לבדוק.

פעולה: ('search(query='Czech Republic currency 2025

תצלפית: התוצאה מראה שצ'כיה משתמשת בكورونא צ'כית.

תשובה: קורונה צ'כית".

▲ אל תעשה: ReAct Unstructured

"מה המطبع של צ'כיה? תחפש ותגיד לי."

✓ עשה: Tree-of-Thoughts

"בעיה: מצא את הדרך הקצרה ביותר בגרף.

רמת 1: 3 אסטרטגיות (A*, DFS, BFS).

רמת 2: כל אסטרטגיה מייצרת 2 פתרונות.

בחר את הפתרון עם הציון הטוב ביותר."

▲ אל תעשה: Single Path

"מצא את הדרכ הקצרה ביותר."

✓ עשה: Role-Based Specific

"אתה פרופסור לכלכלה התנהוגית. הסבר את פרודוקס הבחירה לסטודנטים ב-200

מיליון. השתמש בדוגמה מעולם הקנייה המקוונת."

3.9 חלון ההקשר בפעולה: Context Window Mechanics

מודל שפה לא זוכר. הוא קורא מחדש מוחדר. זו נקודת קריטית. בכל פעם שאתה שולח הודעה, המודל מקבל את כל ההיסטוריה מוחדר. הוא לא "נזכר" במה שאמר קודם. הוא פשוט רואה את זה בחלון הכניסה. ככל שהシיחה מתארכת, החלון מתמלא. ובסוף – הוא נגמר. במקרה זה נראה איך כל שיטה משפיעה על חלון ההקשר.

3.9.1 צמידת החלון: Chain-of-Thought

ב-T-Co, המודל מייצר "טוקנים של חשיבה" שנכנסים לחלון. המשפט "בוא נחשוב עוד אחר עתה" משנה את התנהוגות המודל. במקום לкопץ לתשובה, הוא מייצר שלבי ביןיהם. השלבים האלה תופסים מקום.

שלב 1: השאלה הראשונה (T_1)

טבלה 7: חלון ההקשר ב-T-Co – שלב 1

חלון יציאה	חלון כניסה
לדומה טובות דעך רחא דעך בשחן אוב סירודכ 5: הלחתה 1. $6 = 3 \times 2$: היינק 2. סירודכ $3 + 6 = 11$: לכה זס. סירודכ 9 - 2 = 9: דובייא. 9: הבושתת	תכרעם תייחנה וויטצם תודית רתוף התא שמתשם הנוק אווה. סיינט ירודכ 5 שי ר' גורל סירודכ 3 תחא לכבר טואספוק 2 ?ול שי המכ. 2. דבאמ אווה רגירת דעך רחא דעך זו לע בשחן אוב

שלב 2: שאלת המשך (T_2)

שים לב: חלון הכניסה עכשו מכיל את כל מה שקרה ב- T_1 .

טבלה 8: חלון ההקשר ב-T-Co – שלב 2

חלון יציאה	חלון כניסה
לדומה טובות שי, תמדוקה הבושתת לע סבבתה סירודכ 9 בושית $9 \div 3 = 3$ סירודכ 3 לבקי דלי לפ	[ל"נכ]: תכרעם תייחנה ...סירודכ 5 שי ר' גורל: T_1 שמתשם 9: הבושתת ... בשחן אוב: T_1 לדום שמתשם T_2 : הוש סירודכח תא קלחי ר' גור סא דחא לכ המכ, סידלי 3 זיב הושב לבקי

ReAct 3.9.2: עיצירות וחוויות

ב-ReAct, החלון גדל בקפיצות. המודל עוצר, מבצע פעולה, ומתקבל תוצאה. כל מהזור מוסף שלושה חלקים: Observation, Action, Thought.

טבלה 9: חלון הקשר ב-ReAct – מהזור שלם

חלון יציאה	חלון כניסה
<p>(1 בלש) לדום Thought: הcz ימ אוצמל זירצ נא Super Bowl 2024. Action: Search[“Super Bowl 2024 winner”]</p> <p>[ואכ תרצוע תכרעמה]</p> <p>(2 בלש) לדום Thought: שרפחה תא בשחא וישכע 22. 25. توفח. Answer: 3 שרפחה. וcz ספ'יצה: תודוקן.</p>	<p>תכרעם תייחנה שופיך עונמל השיג דל שי. טמרופב שמתשה Thought: [הבשחט] Action: [הלועפ] Observation: [האוצאות]</p> <p>שמתשם היה המו Super Bowl 2024 -ב הcz ימ ?שרפהה</p> <p>(תכרעמהמ) תיפצת 25-22. וחצין ספ'יצ יטיס סזנק</p>

שים לב: ה-Observation לא נוצר על ידי המודל. הוא מזרק על ידי המערכת החיצונית.
 זה מה שהופך את ReAct לתליי בקוד חיצוני.

Tree-of-Thoughts 3.9.3: ענפים מקבילים

ב-ToT, החלון מתפצל. המודל מייצר מספר אפשרויות, מעריך כל אחת, ובורח. ביחסם מתקדם, זה דורש **מספר קריאות** למודל – אחת לכל ענף.

טבלה 10: חלון הקשר ב-ToT – הערכת ענפים

חלון יציאה	חלון כניסה
<p>לדום וועשה תא סירה שלבה: 1 וויכ אפק סלועהו. Score: 6/10. יdem ילאנג –</p> <p>,”חצnel 12:00 התיה העשה”: 2 וויכ שלבה למלם. Score: 9/10. יתרויאו וركסם –</p> <p>סושו רותפה לע צחל אווה: 3 וויכ הרק אל רבד. Score: 7/10. יטאשילק –</p> <p>וויכ סע דישמן: הטלחה 2.</p>	<p>תכרעם תייחנה החמומם רפוס התא. הלחתהל סינוש סינוויכ 3 עצה 1-10. וויכב דחא לכ זירעה רתוייב חיטבמה תא רחוב</p> <p>שמתשם שלב לע רופיסל החיתוף טפשם בותח ומאזה תא רצועש וועש אצומש</p>

3.9.4 מה קורה כשהחלון נגמר?

זו שאלת קרייטית. לכל מודל יש מגבלת Context Window

: עד 128000 טוקנים - GPT-4

: עד 200000 טוקנים - Claude 3

: עד 1000000 טוקנים - Gemini 1.5

כשהחלון מתמלא, המערכת חייבת למחוק חלק מההיסטוריה. בדרך כלל, המידע הישן ביחס נמחק ראשוני (FIFO). המשמעות: בשיחה ארוכה, המודל "ישכח" את תחילת השיחה. הנחיה המערכת בדרך כלל נשמרת, אבל ההודעות הראשונות – יעלמו.

Context Management

ניהול חלון ההקשר הוא חלק קרייטי מהנדסת פרומפטים. ככל שהשיטה מורכבת יותר (CoT < ReAct < ToT), כך היא צורכת יותר טוקנים. תכנון נכון מażן בין עומק החשיבה לבין הזיכרון הזמן.

3.10 סיכום הפרק: Chapter Summary

למדיינו שלוש תבניות מתקדמיות להנדסת פרומפטים:

.1. **Chain-of-Thought**: מלמד מודלים לחשב בשלבים. Self-Consistency מוסיפה הצבעה בין נתיבים.

.2. **ReAct**: משלב חשיבה עם פעולה. גרסה 2.0 מוסיפה רפלקטיה.

.3. **Tree-of-Thoughts**: חוקר מספר נתיבים במקביל. גיזום חוסך משאבים.

בנוסף, למדיינו על:

- Few-shot learning – 3 דוגמאות הן נקודת האיזון

- Role-based prompting – תפקיד ספציפי עוזר יותר מተפקיד כללי

בפרק הבא נלמד כיצד להשתמש בתבניות אלו לבניית סוכני AI.

4 הנדסת פרוומפטים לסוכני AI

מטרות הלמידה

בסיום פרק זה, הקורא יוכל:

- לעצב פרוומפטים לסוכנים אוטונומיים

- לשלב state management ו-tool-use

- לבנות guardrails ומנגנוני בטיחות

- להבין את עקרונות ה-Agent-Computer Interface

4.1 מבוא: מה המערכת לפועלה

מאז ומתמיד חלם האדם על עוזרים אוטונומיים. מהගולם של פראג ועד רובוטי המדע הבדיוני, החזון היה אחד: ייצור שמבחן הוראות ופועל לפייהן. כיום, חזון זה הופך למציאות. מודלי שפה גדולים (LLMs) כבר לא רק עונים על שאלות. הם יכולים לפעול בעולם האמתי [14].

סוכני AI הם מערכות שמשלבות חשיבה עם פעולה. הם מקבלים משימה, מתכוונים צעדים, ומבצעים אותם. לעיתים הם קוראים לכליים חיצוניים. לעיתים הם מחפשים מידע באינטרנט. תמיד הם צריכים לדעת מתי לעוזר.

אבל יש בעיה מהותית. מודלי שפה לא נבנו לפעולה. הם נבנו לחיזוי המילה הבאה. כדי להפוך אותם לסוכנים, צריך פרוומפטים מיוחדים. פרוומפטים שמגדירים לא רק מה חשוב, אלא גם מה לעשות [20].

4.2 ממתק סוכנו-מחשב: ACI

ecafretnI retupmoC-tnegA

פרוומפטים לסוכנים חייבים לכלול ארבעה רכיבים:

1. תיאורי כלים עם דוגמאות

2. פורמט מצב (state)

3. פרוטוקול טיפול בשגיאות

4. תנאי סיום

כמו שמחשב צריך ממתק משתמש, סוכן צריך ממתק לעולם. זהו ה-Agent-Computer Interface או ACI. ללא ממתק ברור, הסוכן לא יודע מה הכלים שלו. הוא לא יודע מה המצב

הנוכחי. הוא לא יודע מתי לעצור. מחקר Anthropic מ-2024 מראה טובנה מפתיעה [20]. הפתרונות פשוטים עובדים וכי טוב. ארכיטקטורות מורכבות נכשלות. תבניות פשוטות וניתנות להרכבה מצלחות.

ReAct 4.2.1 **תבנית**

תבנית ReAct היא הבסיס לרוב הסוכנים [14]. השם מגיע מ-Reasoning + Acting. הסוכן מבצע מעגל פשוט:

1. **חשוב** (Think): נתח את המצב

2. **פעל** (Act): בחר וקרה לכלי

3. **צפה** (Observe): קרא את התוצאה

4. חוזר לשלב 1 עד להשתת המטרה

ב-2023 הוסיפו שלב רביעי: **הרהר** (Reflect). הסוכן בודק אם הפעולה הייתה אפקטיבית. אם לא, הוא משנה את האסטרטגיה. גרסה זו נקראת ReAct 2.0 [21].

4.3 **ארכיטקטורות סוכנים**: Agent Architectures

טבלה 11 מציגה את הארכיטקטורות הנפוצות. כל אחת מתאימה לסוג אחר של משימות.

טבלה 11: ארכיטקטורות פרומפטים לסוכנים

מתאים ל...	Success Rate	Token Budget	רכיבים	תבנית
שימוש בכלי פשוט	0.78	300-500	Think, Act, Obs	ReAct Classic
שימוש בכלי מורכב	0.85	500-800	+Reflect, Adapt	ReAct 2.0
ריבוי סוכנים	0.88	400+n*200	+ מנהל עובדים	Manager-Worker
ניתוב דינמי	0.82	300+n*150	+ מיון מומחים	Decentralized

כפי שראים בטבלה, התבנית Manager-Worker משגינה את שיעור ההצלחה הגבוה ביותר. אבל היא גם cocci יקרה במונחי tokens. הבחירה תלויות במשימה ובתקציב.

4.4 מוקחות בטיחות כפרומפטים: Guardrails as Prompts

stpmorP sa liardrauG

כל action guardrail הינו פרומפט בפנוי עצמו שרצ' במקביל לסוכן. מבנה: "IF action violates policy THEN raise Exception".

סוכנים יכולים לטעות. הם יכולים לחושף מידע רגיש. הם יכולים לבצע פעולות מסוכנות. לכן צריך מוקחות בטיחות. NVIDIA פיתחה את NeMo Guardrails [22]. זו ספרייה שמוסיפה שכבות הגנה לסוכנים. כל שכבה היא פרומפט עצמאי. הפרומפטים רצים במקביל לסוכן. "שנים ארבעה סוגי עיקריים של guardrails:

1. **סינון PII:** מזהה מידע אישי רגיש

2. **זיהוי Jailbreak:** מזהה ניסיונות לעקוף הגבלות

3. **בטיחות כלים:** מדריך סיכון של כל פעולה

4. **RELONAVIOT:** בודק אם התשובה קשורה לשאלת

טבלה 12 מפרטת את סוגי ה-guardrails ואת הפרומפטים שלהם.

טבלה 12: סוגי מוקחות בטיחות ופרומפטים

סוג	דוגמת פרומפט	תנאי הפעלה	פעולה בהפירה
סינון PII	סרוק output תעבור: ת.ז., כרטיס אשראי regex התאמת	RegularExpression	מיסוך + רישום
Zero Knowledge	אם המקרה לא מנסה להשוו sys prompt ?system prompt prob > 0.7	prob > 0.7	+ חסימה התראה
בטיחות כלים	דרג סיכון: read=1, write=2, delete=3 Risk > 2	Risk > 2	אישור אנושי
RELONAVIOT	אם המקרה קשור לשאלת? output-hash == hash CoT similarity < 0.6	similarity < 0.6	נסה שוב עם CoT

4.5 מטא-פרומפטינג: Meta-Prompting

gnitpmorP-ateM

פרומפט שכותב פרומפטים. ב-2024 הוכח ש-meta-prompt עם 2 דוגמאות מגדיל את שיעור ההצלחה של סוכנים ב-34% [23].

מה אם במקומות לכתוב פרומפטים ידנית, נבקש מה-LLM לכתוב אותן? זו הרעיון מאחרוי Meta-Prompting. פרומפט שמייצר פרומפטים. Kalai-1 Suzgun הראו ב-2024 שהגישה עובדת [23]. הם פיתחו מסגרת שנקראת Task-agnostic Scaffolding. המסגרת מותמכת במבנה המשימה, לא בתוכן. התוצאות היו מרשימות. מודל Qwen-72B עם meta-prompt בלבד השיג תוצאות state-of-the-art.:Meta-Prompting היתרונות של

1. **יעילות tokens:** פחות tokens נדרשים

2. **השוויה הוגנת:** פחות תלות בדוגמאות ספציפיות

3. **יכולת zero-shot:** עובד ללא דוגמאות

4.6 נוסחאות מתמטיות: Mathematical Formulas

הנת הביצועים של סוכנים דורשת מודלים מתמטיים. שלוש משוואות מרכזיות מגדירות את ההצלחה של סוכן.

משואה 1.4: שיעור ההצלחה סוכן

$$(11) \quad \text{Success} = P(\text{correct action}) \cdot \prod_{i=1}^T P(\text{observation}_i \text{ valid}) \cdot P(\text{termination})$$

משואה 11 מראה שההצלחה היא מכפלה של שלושה גורמים:

- $P(\text{correct action})$: הסתברות לבחור פעולה נכונה

- $\prod_{i=1}^T P(\text{observation}_i \text{ valid})$: הסתברות שכל התוצאות תקיןות

- $P(\text{termination})$: הסתברות לעצור בזמן הנכון

כאשר T הוא מספר הצעדים. שימו לב: אם סוכן עושה הרבה צעדים, הסתברות הכוללת יורדת. לכן סוכנים טובים ממזערים את מספר הצעדים.

משואה 2.4: יעילות MgnitpmorP-ateM

$$(12) \quad \text{Eff}_{\text{meta}} = \frac{\text{Acc}_{\text{meta-generated}} - \text{Acc}_{\text{human-written}}}{\text{Acc}_{\text{human-written}}} \times 100\%$$

משואה 21 ממחשבת את השיפור שambil Meta-Prompting. החוקרים מראים $\text{Eff}_{\text{meta}} = 34\% \pm 5\%$ עבור סוכנים חדשים [23].

משוואה 3.4: ביטחון בבחירה כל

$$(13) \quad \text{Select}(t) = \frac{\exp(\text{sim}(q, d_t))}{\sum_{j=1}^M \exp(\text{sim}(q, d_j))} \cdot \mathbb{1}[\text{guardrail}(t) = \text{pass}]$$

משוואה 31 מגדירה איך סוכן בוחר כל. המשוואה משלבת שני גורמים:

- **דמיון סמנטי:** $\text{sim}(q, d_t)$ בין השאלה q לתיאור הכל d_t

- **מעבר גדר:** $\mathbb{1}[\text{guardrail}(t) = \text{pass}]$

שימוש לב: כל שוחס על ידי guardrail מקבל ציון אפס. זה מבטיח בטיחות גם כשהכל נראה רלוונטי.

4.7 שימוש בכלים: Tool-Use

היכולת להשתמש בכלים היא מה שمبادיל סוכנים מ-chatbots. הראה שמודלים יכולים ללמידה להשתמש בכלים עצמם [24]. ToolLLM הרחיב את זה ל-16000 APIs [25]. Gorilla הראה איך לחבר מודלים ל-APIים מסוימים [26]. OpenAI הציגה ב-2024 את Structured Outputs [27]. זה מבטיח שהפלט תואם לסכמה מוגדרת מראש. ב-evaluations מורכבים, המודל החדש השיג 100% דיק. לעומת זאת, GPT-4 השיג פחות מ-40%.

Berkeley Function-Calling Leaderboard מודד את היכולת הזו [28]. הוא בודק:

- קריאות מרובות לפונקציות

- קריאות מקבילות

- קריאות רב-שלביות

4.8 דוגמאות עשה ולא תעשה: Do and Don't Examples

הבדל קטן בפרומפט יכול לעשות הבדל גדול בביטויים. להלן דוגמאות מעשיות.

✓ **עשה:** Tool Specification

"כל": search_database

תיאור: מhapus בסיס נתונים ללקוחות.

פרמטרים: .query (string, example: 'age > 30 AND city="TLV")", limit (int, default=10)

מחזיר: List[Customer] עם שדות: ".id, name, age"

▲ **אל תעשה:** Vague Tool

"כל": .search. תשתמש בו לחיפוש".

ההבדל ברור: הפרומפט הטוב מספק דוגמאות ופרמטרים. הפרומפט הגרוע לא מסביר כלום.

✓ עשה: State Management

.State: {'step': 2, 'last_action': 'search', 'results': [...], 'goal': 'find premium customers'}
בהתבסס על ה-state, בחר את הפעולה הבאה.

▲ אל עשה: No State

"תמשיך לחפש".

✓ עשה: Guardrail as Prompt

user.status == 'inactive' AND request.approval == 'delete_user', בדוק:
אם הפעולה היא 'delete_user', בדוק:
אחרת, דחה את הבקשה. True

▲ אל עשה: No Guardrails

"תמחק משתמשים לפי הצורך".

✓ עשה: Meta-Prompting

"אתה מומחה לכתיבת פרומפטים. כתוב פרומפט ל-summarization של דוחות כספיים.
הפרומפט חייב לכלול: (1) תפקיד, (2) פורמט, (3) מבדדים חשובים. דוגמה: ..."

▲ אל עשה: Direct Prompt

"כתב פרומפט לסיכום דוחות".

4.9 סיכום הפרק: Chapter Summary

בפרק זה למדנו כיצד לעצב פרומפטים לשוכני AI. הנקודות המרכזיות הן:

1. **משמעות ACI:** כל סוכן צריך ארבעה רכיבים: כלים, מצב, טיפול בשגיאות, ותנאי סיום.
2. **מבנה ReAct:** מעגל של חשיבה-פעולה-תצפית הוא הבסיס לרוב הסוכנים.
3. **מעקות בטיחות:** כל guardrail הוא פרומפט עצמאי שרצ במקביל.
4. **פרומפטים שכותבים פרומפטים:** Meta-Prompting יכולם לשפר ביצועים ב-34%.
5. **שימוש בכלים:** תיאור מדויק של כלים קריטי להצלחה.

בפרק הבא נלמד על הערכה, אופטימיזציה והטמעה בייצור.

5 הערבה, אופטימיזציה והטמעה ביצור

מטרות הלמידה

בסיום פרק זה, הקורא יוכל:

- ליישם מетодולוגיית A/B testing לפרויקטים
- לנוהל CI/CD וversion control לפרויקטים
- לבצע מעקב (monitoring) וזיהוי drift
- להבין את עקרונות האופטימיזציה לעולות-יעילות

5.1 מבוא: מההערכה ליצור

לפני הכתב, בני האדם סמכו על האזכור. כל סיפור שהועבר מפה לאוזן השתנה מעט. גרסה אחת הפכה לעשר. איש לא ידע מה היה המקור. הנדסת פרויקטים עוברת מהפכה דומה. מה שהתחילה כאמונות אישית הופך לתהיליך תעשייתי. המעבר הזה דורש כלים חדשים: ניהול גרסאות, בדיקות מובהקות, וזיהוי סחיפת. ללא הכלים האלה, פרויקט שעבודת היום עלול להיכשל מחר. מחקר של Booking.com בחן 150 מודלים מוצלחים [29]. הם גילו עובדה מפתיעה: שיפור בדיקת המודל לא תמיד מוביל לשיפור עסקיו. הדרך היחידה לדעת אם שינוי עובד היא לבדוק אותו ביצור. זה מחייב ניסויים מבוקרים, לא תחשות בטן.

5.2 ניהול גרסאות פרויקטים: Prompt Versioning

gninoisreV tpmorP

כל פרויקט חייב לכלול:

.1 Version ID סמנטי (X.Y.Z)

.2 Test set hash – טביעה אצבע של נתוני הבדיקה

.3 מדדי ביצוע: דיקוק, זמן תגובה, עולות

.4 תאrik הטמעה ושם הכותב

בתוכנה מסורתית, קוד הוא דטרמיניסטי. אותו קוד נותן אותה תוצאה. בפרויקטים, המצב שונה לחלוטין. מודל שפה אינו דטרמיניסטי. הוא יכול לתת תשובות שונות לאותו פרויקט.

לכן ניהול גרסאות בפרומפטים מורכב יותר. נדרש לעקוב לא רק אחרי הטקסט, אלא גם אחרי הביצועים. כל גרסה חייבת לכלול מטא-דאטा מפורט. זה מאפשר לחזור לאחרה שימושה משותבש.

הגישה המומלצת היא Semantic Versioning [30]. גרסה ראשית (X) משתנה כשהפרומפט משתנה באופן משמעותי. גרסה שנייה (Y) משתנה כמשמעותם יכולות. גרסה קטנה (Z) משתנה כמשמעותם באגים.

5.3 מובהקות סטטיסטיות: Statistical Significance

ecnacifingiS lacitsitatS

לשינויים בפרומפט, דרוש גודל מוגן מינימלי. ל-95% ביטחון ו-80% עוצמה, נדרשים לפחות 1000 דוגמאות לכל קבוצה.

המחפה המדעית לימדה אותנו דבר חשוב. תחשות בטן מטעות. רק ניסויים מבוקרים יכולים להבדיל בין אמת לאשלה. כמשנים פרומפט, קל לחשב שהוא השתרף. אבל ייתכן שהשיפור הוא מקרי. כדי לדעת בודדות, נדרש מספיק דוגמאות. הנוסחה לחישוב גודל המוגן היא:

משוואה 5.1: חישוב גודל מוגן

$$(14) \quad n = \frac{(z_{1-\alpha/2} + z_{1-\beta})^2 \cdot (\sigma_1^2 + \sigma_2^2)}{\Delta^2}$$

במשוואה זו:

- $z_{1-\alpha/2}$ הוא ערך קרייטי לרמת המובהקות (1.96) ל-95%

- $z_{1-\beta}$ הוא ערך קרייטי לעוצמה (0.84) ל-80%

- σ הוא סטיית התקן של המודד

- Δ הוא ההבדל המינימלי שרצחים לאזהות

מחקר של Apple [31] הרחיב את הנוסחאות הללו. הם פיתחו שיטות לנוטונים מתואימים. זה חשוב במיוחד כמשתמשים חוזרים מספר פעמים.

5.4 מתודולוגיות בדיקות A/B: A/B Testing Methodology

הניסוי המבוקר האקדמי היה המתנה הגדולה של הרפואה למדע. הוא אפשר להבדיל בין תרופות אמיתיות לפלאזבו. בהנדסת פרומפטים, A/B testing, ממלא תפקיד דומה.

בניסוי A/B, מחלקים את המשתמשים לשתי קבוצות. קבוצת הביקורת מקבלת את הפרומפט החדש. קבוצת הטיפול מקבלת את הפרומפט החדש. משווים את הביצועים ובודקים אם ההבדל מובהק.

טבלה 13: מетодולוגיות בדיקות A/B לפרומפטים

ההחלטה	p-value	מדד הצלחה	משך	וריאנט
בסיס	-	0.891	7 ימים	Control (v1.2)
פרסום	0.003	0.905	7 ימים	Treatment A
פרסום (מנצח)	0.001	0.912	7 ימים	Treatment B (CoT)
דיכוי	0.089	0.898	7 ימים	Treatment C

טבלה 13 מציגה דוגמה לניסוי עם ארבעה וריאנטים. רק וריאנטים עם p-value קטן מ-0.05 נחשבים מובהקים. וריאנט C נדחה למינות שחדוק שלו גבוה מהבסיס. הסיבה: ההבדל לא מובהק סטטיסטית. איור 8 מציג את תוצאות הניסוי לאורץ זמן. הקו האדום מייצג את הפרומפט החדש (v1.3-cot). הקו הכחול מייצג את הפרומפט הישן (v1.2). ההבדל עקבי לאורץ כל הימים מה שמחזק את המסקנה.

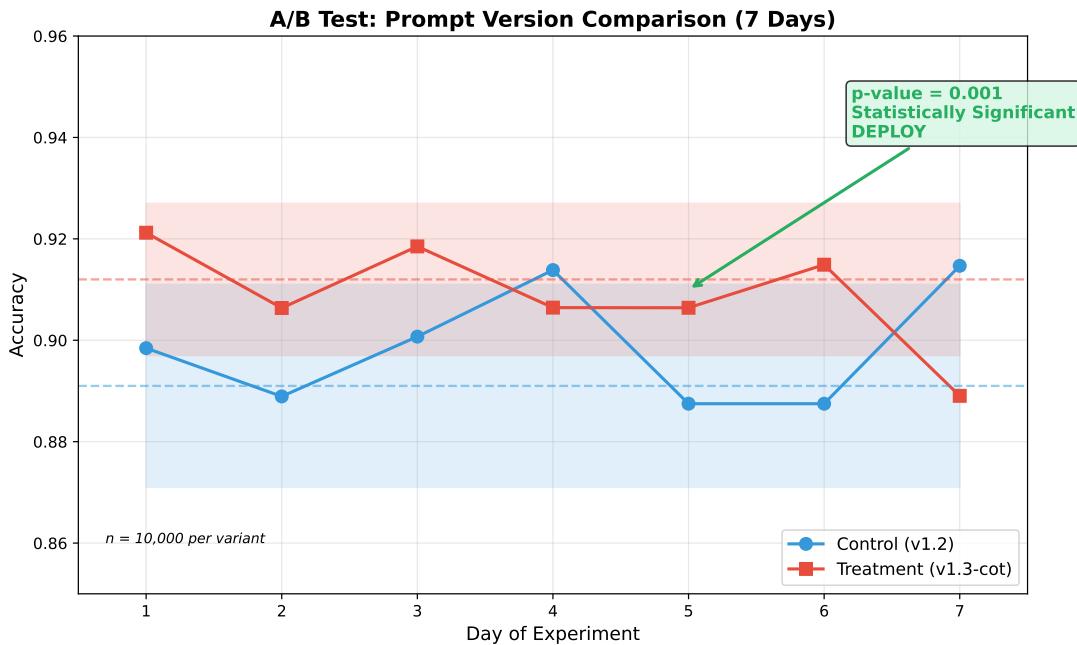
5.5 זיהוי סחיפה: Drift Detection

noitceteD tfirD

KL divergence מודד את המרחק בין שתי התפלגויות. אם $D_{KL} > 0.15$, יש להפעיל התראה ולבדק את הפרומפט.

כל המערכות נוטות לאי-סדר. זה לא פילוסופיה – זה החוק השני של התרמודינמיקה. גם מערכות AI סובלות מטופעה דומה. עם הזמן, הנתונים משתנים. משתמשים חדשים מתנהגים אחרת מהישנים. העולם משתנה, והמודל לא מודע לכך. זהה "סחיפה" (drift). מבחן של Allali ו-Kurian [32] הציג שיטה לזיהוי סחיפה. הם השתמשו ב-KL divergence כמדד:

משוואת 5.2: זיהוי סחיפה באמצעות KL Divergence



איור 8: תוצאות ניסוי B/A: השוואת גרסאות פרומפט למשך 7 ימים

$$(15) \quad D_{KL}(P_{out} || P_{train}) = \sum_x P_{out}(x) \log \frac{P_{out}(x)}{P_{train}(x)}$$

במשוואה זו:

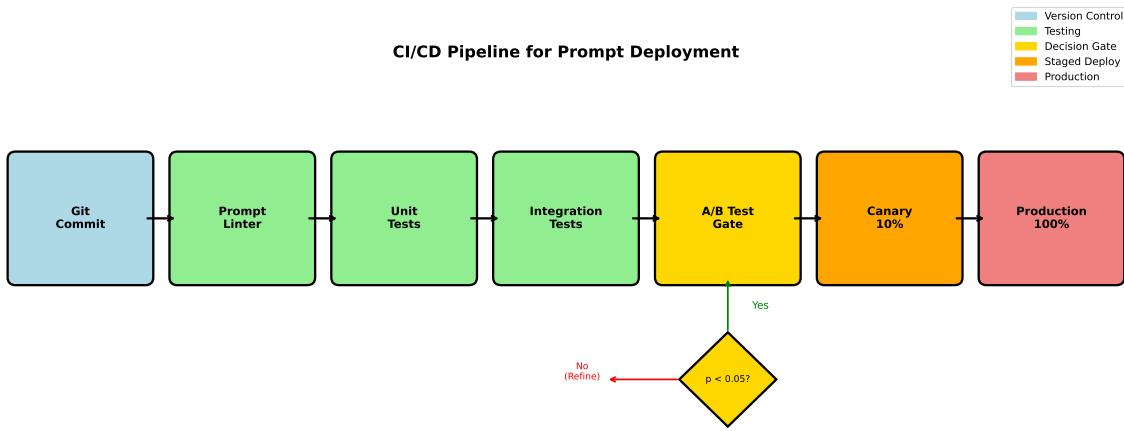
- P_{out} היא התפלגות הפלטים ביצור
- P_{train} היא התפלגות הפלטים בזמן האימון
- ערך תמיד חיובי או אפס
- ערך גבוה מעיד על סחיפה

מחקר של CMU [33] בוחן שיטות זיהוי ביצור. הם מצאו ש- KL divergence נותנת את מיידי כשמתרחשת סחיפה. אבל הוא רגש לערכיהם קיצוניים. לכן כדאי לשלב אותו עם מדדים נוספים כמו Jensen-Shannon divergence.

5.6 צינור CI/CD לפרומפטים: CI/CD Pipeline

האוטומציה שחררה את בני האדם מעבודה חוזרת. היא מאפשרת להתמקד ביצירתיות. בהנדסת פרומפטים, CI/CD ממלא תפקיד דומה. איור 9 מציג את הצינור המומלץ. הוא כולל שבעה שלבים:

- שמירת הפרומפט במערכת ניהול גרסאות .1
- בדיקת תחביר ותבניות .2



איור 9: צינור CI/CD לפרוומפטים: משלב הפיקוח ועד לייצור

.3 – בדיקות יחידה על דוגמאות Unit Tests .

.4 – בדיקות אינטגרציה עם המערכת Integration Tests .

.5 – שער החלטה: האם $p > 0.05$? – A/B Test Gate .

.6 – פרישה לחלק קטן מהמשתמשים 10% Canary .

.7 – פרישה מלאה 100% Production .

מחקר של מנהיר ארבעה עקרונות ל-MLOps [34] Google Cloud (Continuous Integration- CI) ,(Continuous Monitoring) CM-1 ,(Continuous Training) CT ,(Continuous Delivery) CD ,tion) כל העקרונות הללו רלוונטיים גם לפרוומפטים.

5.7 אופטימיזציה עלות-דיוק: Cost-Accuracy Optimization

הכללה מלמדת שלכל בחירה יש מחיר. ב-AI, אנחנו סוחרים דיוק בזמן תגובה, איקות בעלות. השאלה היא: מה האיזון הנכון? טבלה 14 מציגה את המדרדים החשובים. לכל מדד יש סף מומלץ ותנאי התראה. כשתנאי ההתראה מתקיים, צריך לבדוק את הפרוומפט. המושג "חיזית פארטו" (Pareto frontier) מ塔אר את האיזון האופטימלי [35]. כל נקודה על החיזית היא פתרון שלא ניתן לשפר בו אחד מבלי לפגוע באחר.

5.3.3 מדדיעילות Pareto

$$(16) \quad \text{Efficiency} = \frac{\text{Accuracy}}{\log(\text{Cost} + 1)} + \lambda \cdot \text{Latency}^{-1}$$

במושוואה הזו:

- Accuracy – הוא הדיק של הפרוומפט

טבלה 14: מדריך הערכה לפרומטטים ביצור

תנאי התראה	ס'	נוסחה	מדד
ירידה < 2% ב- 24 שעות	> 0.90	TP/(TP+FN)	דיוק (Accuracy)
עלייה < 1%	< 0.05	FP/(FP+TN)	שיעור ה以习近平
< 2500 מ"ש	< 2000ms	percentile(99)	זמן תגובה P99
0.015\$ <	< \$0.01	tokens □ price	עלות למשימה
0.15 <	< 0.10	D_{KL}	ציון סחיפה

- Cost - היא העלות לבקשה

- Latency - הוא זמן התגובה

- λ הוא משקל שמאזן את חישיבות זמן התגובה

מחקר של ParetoQ [36] הראה שאפשר לשפר את חזית פארטו. הם מצמכו את הפער בין דחיסה ל-2 ביט לבין דיוק מלא. העיקרון דומה: חפשו פתרונות שיפורים את הייעילות הכלולית.

Do and Don't Examples: עשה ולא תעשה 5.8

✓ **עשה:** Version Control

"פרומפט": v1.4-cot-manager

Tests: test_set_gsm8k_hash=0x4f2a1c

Metrics: accuracy=0.912, latency_p99=1850ms, cost=\$0.009

Deployed: 2025-12-02 06:34 UTC

"Author: yoram.segal

▲ **אל תעשה:** No Versioning

"פרומפט": גרסה אחרונה"

✓ **עשה:** Statistical Significance

"הגדرتית גודל מדגם $n = 1000$, אזי $p\text{-value} = 0.003 < 0.05$, אז מפרנס את הפרומפט החדש".

▲ **אל תעשה:** No Testing

"אני מרגיש שהפרומפט החדש טוב יותר, אז אני מפרנס אותו."

✓ **עשה:** Drift Monitoring

"בודק KL divergence בכל 6 שעות. אטמול היה $0.08 < 0.18$, היום $0.15 < 0.18$. מעדכן את הפרומפט".

▲ **אל תעשה:** No Monitoring

"הפרומפט עובד, לא צריך לבדוק".

✓ **עשה:** Cost Optimization

"גרסה cot-1.3 עולה \$0.009 בלבד. בדקתי עם CoT בעלות \$0.007, דיקון רק 2% נמוך. מעבר ל- cot-1.3 ביצור".

▲ **אל תעשה:** Ignore Cost

"דיקון הכי חשוב, לא משנה הוצאות".

5.9 סיכום הפרק: Chapter Summary

בפרק זה למדנו כיצד להעביר פרומפטים מהמעבדה לייצור. הנה העקרונות המרכזיים:

1. **ניהול גרסאות:** כל פרומפט צריך ID, מדדי ביצוע, ותאריך הטמעה.
2. **МОВЕКАТОТ СТТИСТИЧИХ:** אל תסמכו על תוצאות. השתמשו במדגם של לפחות 1000 דוגימות לכל קבוצה.
3. **בדיקות B/A:** השוו את הפרומפט החדש לשן. פרסמו רק אם $p\text{-value} > 0.05$.
4. **זיהוי סחיפה:** עקבו אחרי KL divergence. התיריעו כ- $0.15 > D_{KL}$.
5. **צינור CI/CD:** אוטומציה של כל התהילה, משלב הפיתוח ועד לייצור.
6. **אופטימיזציית עלות-דיקון:** חפשו את נקודת האיזון על חיזית פארטו.

המסר המרכזי: הנדסת פרומפטים ביצור היא מדע, לא אומנות. היא דורשת מדידה, בדיקות, ואיתרציה מתמדת. רק כך נוכל לבנות מערכות אמינות שעובדות למשך זמן.

5.11 English References

- 1 P. Sahoo, A. K. Singh, S. Saha, V. Jain, S. Mondal, and A. Chadha, “A systematic survey of prompt engineering in large language models: Techniques and applications,” *arXiv preprint arXiv:2402.07927*, 2024.
- 2 L. Huang et al., “A survey on hallucination in large language models: Principles, taxonomy, challenges, and open questions,” *arXiv preprint arXiv:2311.05232*, 2024.
- 3 S. Tonmoy, S. Zaman, V. Jain, A. Rani, V. Rawber, and A. Chadha, “A comprehensive survey of hallucination mitigation techniques in large language models,” *arXiv preprint arXiv:2401.01313*, 2024.
- 4 C. E. Shannon, “A mathematical theory of communication,” *Bell System Technical Journal*, vol. 27, no. 3, 379–423, 1948.
- 5 N. F. Liu et al., “Lost in the middle: How language models use long contexts,” *Transactions of the Association for Computational Linguistics*, vol. 12, 157–173, 2024. doi: [10.1162/tacl_a_00638](https://doi.org/10.1162/tacl_a_00638) [Online]. Available: <https://arxiv.org/abs/2307.03172>
- 6 S. Schulhoff et al., “The prompt report: A systematic survey of prompting techniques,” *arXiv preprint arXiv:2406.06608*, 2024.
- 7 A. Vaswani et al., “Attention is all you need,” *Advances in Neural Information Processing Systems*, vol. 30, 5998–6008, 2017. [Online]. Available: <https://arxiv.org/abs/1706.03762>
- 8 J. Wei et al., “Chain-of-thought prompting elicits reasoning in large language models,” *Advances in Neural Information Processing Systems*, vol. 35, 24824–24837, 2022. [Online]. Available: <https://arxiv.org/abs/2201.11903>
- 9 T. Khot et al., “Decomposed prompting: A modular approach for solving complex tasks,” in *International Conference on Learning Representations (ICLR)*, 2023. [Online]. Available: https://openreview.net/forum?id=_nGgzQjzaRy
- 10 X. Wang et al., “Self-consistency improves chain of thought reasoning in language models,” *arXiv preprint arXiv:2203.11171*, 2023.
- 11 S. Diao, P. Wang, Y. Lin, R. Pan, X. Liu, and T. Zhang, “Active prompting with chain-of-thought for large language models,” in *Proceedings of the 62nd Annual Meeting of the ACL*, 2024, 1330–1350. [Online]. Available: <https://aclanthology.org/2024.acl-long.73/>

- 12 W. Chen, Y. Liu, and H. Zhang, “The few-shot dilemma: Over-prompting large language models,” *arXiv preprint arXiv:2509.13196*, 2025. [Online]. Available: <https://arxiv.org/abs/2509.13196>
- 13 A. Madaan et al., “Self-refine: Iterative refinement with self-feedback,” *Advances in Neural Information Processing Systems*, vol. 36, 2023. [Online]. Available: <https://arxiv.org/abs/2303.17651>
- 14 S. Yao et al., “React: Synergizing reasoning and acting in language models,” *arXiv preprint arXiv:2210.03629*, 2023.
- 15 S. Yao et al., “Tree of thoughts: Deliberate problem solving with large language models,” *arXiv preprint arXiv:2305.10601*, 2023.
- 16 A. Kong et al., “Better zero-shot reasoning with role-play prompting,” *arXiv preprint arXiv:2308.07702*, 2023. [Online]. Available: <https://arxiv.org/abs/2308.07702>
- 17 M. Zheng, J. Pei, and D. Jurgens, “When a helpful assistant is not really helpful: Personas in system prompts do not improve performances of large language models,” *arXiv preprint arXiv:2311.10054*, 2023. [Online]. Available: <https://arxiv.org/abs/2311.10054>
- 18 J. Kim, N. Yang, and K. Jung, “Persona is a double-edged sword: Enhancing the zero-shot reasoning by ensembling the role-playing and neutral prompts,” *arXiv preprint arXiv:2408.08631*, 2024. [Online]. Available: <https://arxiv.org/abs/2408.08631>
- 19 T. Brown et al., “Language models are few-shot learners,” *Advances in Neural Information Processing Systems*, vol. 33, 1877–1901, 2020.
- 20 Anthropic. “Building effective agents.” [Online]. Available: <https://www.anthropic.com/engineering/building-effective-agents>
- 21 R. Team, “React 2.0: Enhanced reasoning and acting with reflection,” *arXiv preprint*, 2025.
- 22 T. Rebedea, R. Dinu, M. Sreedhar, C. Parisien, and J. Cohen, “Nemo guardrails: A toolkit for controllable and safe llm applications with programmable rails,” *arXiv preprint arXiv:2310.10501*, 2023.
- 23 M. Suzgun and A. T. Kalai, “Meta-prompting: Enhancing language models with task-agnostic scaffolding,” *arXiv preprint arXiv:2401.12954*, 2024.
- 24 T. Schick et al., “Toolformer: Language models can teach themselves to use tools,” in *Advances in Neural Information Processing Systems*, 36, 2023.
- 25 Y. Qin et al., “Toolllm: Facilitating large language models to master 16000+ real-world apis,” *arXiv preprint arXiv:2307.16789*, 2023.

- 26 S. G. Patil, T. Zhang, X. Wang, and J. E. Gonzalez, “Gorilla: Large language model connected with massive apis,” *arXiv preprint arXiv:2305.15334*, 2023.
- 27 OpenAI. “Introducing structured outputs in the api.” [Online]. Available: <https://openai.com/index/introducing-structured-outputs-in-the-api/>
- 28 F. Yan et al., “Berkeley function-calling leaderboard,” *arXiv preprint arXiv:2402.15491*, 2024.
- 29 L. Bernardi, T. Mavridis, and P. Estevez, “150 successful machine learning models: 6 lessons learned at booking.com,” in *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, ACM, 2019, 1743–1751. doi: [10.1145/3292500.3330744](https://doi.org/10.1145/3292500.3330744)
- 30 R. Kumar and P. Sharma, “End-to-end mlops for scalable model deployment,” *International Journal of Computer Trends and Technology*, vol. 72, no. 11, 118–125, 2024. doi: [10.14445/22312803/IJCTT-V72I11P118](https://doi.org/10.14445/22312803/IJCTT-V72I11P118)
- 31 Apple Machine Learning Research, “All about sample-size calculations for a/b testing: Novel extensions and practical guide,” *Apple Machine Learning Research*, 2024. [Online]. Available: <https://machinelearning.apple.com/research/sample-size-calculations>
- 32 J. F. Kurian and M. Allali, “Detecting drifts in data streams using kullback-leibler divergence measure for data engineering applications,” *Journal of Data, Information and Management*, vol. 6, 207–216, 2024. doi: [10.1007/s42488-024-00119-y](https://doi.org/10.1007/s42488-024-00119-y)
- 33 S. Ackermann, M. Bhardwaj, I. Grosse, and K. Hildebrand, “Augur: A step towards realistic drift detection in production ml systems,” in *Workshop on Software Engineering for Responsible AI (SE4RAI)*, CMU SEI, 2022. [Online]. Available: https://www.sei.cmu.edu/documents/614/2022_019_001_877199.pdf
- 34 Google Cloud. “Mlops: Continuous delivery and automation pipelines in machine learning.” [Online]. Available: <https://cloud.google.com/architecture/mlops-continuous-delivery-and-automation-pipelines-in-machine-learning>
- 35 H. A. R. Team, “Pareto-optimized open-source llms for healthcare via context retrieval,” *arXiv preprint arXiv:2409.15127*, 2024.
- 36 Q. R. Team, “Paretoq: Improving scaling laws in extremely low-bit llm quantization,” *arXiv preprint arXiv:2502.02631*, 2025.