

מדריך למשתמש בשירות DSPy

תוכן העניינים

1	חלק א': רקע והבנת הבסיס	
3	1. מהם מודלי שפה גדולים (LLMs) ?
3	2. מהו פרומפט (Prompt)?
3	3. הבעיה: כתיבת פרומפטים היא אמנות
4	4. מהו DSPy?
4	5. מהו אופטימיזר?
2	חלק ב': GEPA - האופטימיזר המתקדם	
4	6. מהו GEPA?
5	7. איך GEPA עובד - הסבר מפורט
5	8. למה הרפלקציה כל כך חשובה?
3	חלק ג': פרמטרים של GEPA	
6	9. פרמטרי תקציב (Budget Parameters)
6	10. פרמטרי מיזוג (Merge Parameters)
6	11. פרמטרי הערכתה (Evaluation Parameters)
7	12. פרמטרי מודל הרפלקציה (LM Reflection Model)
4	חלק ד': מושגי יסוד בהגדרת משימה	
7	13. Signature (חתימה)
7	14. Dataset (сет נתונים)
7	15. Metric (מדד) - דרישות מיוחדות ל-GEPA

9	5. חלק ה': זרימת העבודה המלאה		
9	16. סקירת התהילה	5.1
9	17. מבנה הבקשה לשרת	5.2
10	18. רשימת נקודות קצה (API Endpoints)	5.3
10	19. ניטור התקדמות	5.4
11	6. חלק ו': טיפים ושיטות עבודה מומלצות		
11	20. טיפים להכנת נתוניים	6.1
11	21. טיפים לכתיבת מדדים ל-GEPA	6.2
11	22. פתרון בעיות נפוצות	6.3
12	7. חלק ז': מילון מונחים		

1. חלק א': רקע והבנת הבסיס

1.1. מהם מודלי שפה גדולים (LLMs)?

מודלי שפה גדולים (Large Language Models) הם תוכנות מחשב מתקדמות שעברו אימון על כמויות עצומות של טקסט מה האינטרנט, ספרים ומסמכים. דוגמאות מוכחות כוללות את Claude-1, Gemini, ChatGPT ועוד.

מה הם יודעים לעשות:

- * לענות על שאלות בשפה טבעית
- * לכתוב טקסטים, סיכומים ותרגומים
- * לנתח מידע ולהסיק מסקנות
- * לפתור בעיות לוגיות ומתמטיות
- * לעזור בכתיבת קוד

1.2. מהו פרומפט (Prompt)?

פרומפט הוא הוראה או השאלה שנחנו שולחים למודל השפה. איכות הפרומפט משפיעה באופן דרמטי על איכות התשובה.

דוגמה לפרומפט פשוט: "סכם את הכתיבה הבאה."

דוגמה לפרומפט מפורט: "אתה אנליסט פיננסי בכיר. סכם את הכתיבה הבאה ב-3 נקודות מפתח, תוך התמקדות ברוחים הרבעוניים והתחזית לשנה הבאה. שמור על טון מקצועי ותמציתי."

הפרומפט השני ייתן תשובה מכוκדת יותר כי הוא מגדר:

- * את תפקיד המודל ("אנליסט פיננסי")
- * את המיקוד ("רוחים ותחזית")
- * את הפורמט והטון ("3 נקודות, מקצועי")

1.3. הבעיה: כתיבת פרומפטים היא אמנות

כתבת פרומפטים אפקטיביים (Prompt Engineering) היא משימה מורכבת:

- * **זמן:** נדרשים ניסויים רבים כדי למצוא את הניסוח האופטימלי.
- * **מומחיות:** דורש הינה של אין מודלים מפרשים הוראות.
- * **עקביות:** קשה לשמור על ביצועים עקובים.

* **סקיילביליות:** מה שעוזב למשימה אחת לא בהכרח יעבד לאחרת.

הפתרון של DSPy: כאן נכנס DSPy לתמונה: במקומם לכתוב פרומפטים ידנית, אנחנו מגדירים את המשימה ונוננים לאלגוריתם אופטימיזציה לכתוב את הפרומפט הטוב ביותר לנו!

1.4. מהו DSPy ?

DSPy (ראשי תיבות של Self-improving Python) היא תוכנה שפותחה באוניברסיטת סטנפורד ומאפשרת לתוכנת מודלי שפה בצורה שיטתייה.

העיקרון המרכזי: במקומם לכתוב פרומפטים, אתה מגדיר:

1. **מה הקלט** - איזה מידע המודל מקבל
2. **מה הפלט** - מה אנחנו רוצים שהמודל יחזיר
3. **איך מודדים הצלחה** - מה נחשב לתשובה טובה

לאחר מכן, אלגוריתם אופטימיזציה לומד מדוגמאות ומיציר את הפרומפט האופטימלי אוטומטית.

1.5. מהו אופטימיזר?

אופטימיזר הוא אלגוריתם שמנסה למצוא את ההגדרות הטובות ביותר למשימה מסוימת. בהקשר של DSPy, האופטימיזר:

1. מקבל דוגמאות של קלט ופלט רצוי
2. מנסה וריאציות שונות של פרומפטים
3. מודד את הצלחה של כל וריאציה
4. לומד מהניסיונות ומשתפר
5. מחזיר את הגרסה הטובה ביותר

2 חלק ב': GEPA - האופטימיזר המתקדם

2.1. מהו GEPA ?

GEPA (ראשי תיבות של Genetic-Pareto) הוא אופטימיזר מתקדם ב-DSPy שמשתמש ברפלקציה אבולוציונית לשיפור פרומפטים. מה שמייחד את GEPA הוא שבמוקם להסתמך על תגמולים מספריים בלבד (כמו Reinforcement Learning), הוא מנטח את כל מסלול הביצוע (execution traces) - כולל נתיבי חישיבה, פלטי כלים, ואפילו שניאות - ומשתמש בתובנות אלו לאבולוציה של פרומפטים טובים יותר.

2.2 7. איך GEPA עובד - הסבר מפורט

GEPA פועל בלולאה איטרטיבית של שיפור מתמיד (מחזור העבודה):

1. שלב 1: אתחול (Initialization)

- * מתחילה עם פרומפט בסיסי (baseline).
- * יוצר אוכלוסייה ראשונית של מועמדים (candidates).
- * כל מועמד הוא וריאציה שונה של הפרומפט.

2. שלב 2: הערכתה (Evaluation)

- * כל מועמד מורץ על סט האימון.
- * המערכת אוספת: ציוניים מספוריים (0-1), משוב טקסטואלי מפורט, מסלולי ביצוע (traces), וشنיאות.

3. שלב 3: רפלקציה (Reflection) - הלב של GEPA

- * מודל הרפלקציה (Im_reflection) מנתח את התוצאות.
- * שואל: "למה פרומפט X הצליח פרומפט Y נכשל?"
- * מצאה דפוסים (אילו ניסוחים עובדים טוב יותר) ומפיק תובנות לשיפור.
- * **זה מה שմבדיל את GEPA: הוא לא רק מודד - הוא מבין!**

4. שלב 4: אבולוציה (Evolution)

- * ה-instruction_proposer מציע פרומפטים חדשים המשלבים את התובנות מהרפלקציה.
- * משתמש בטכניקות כמו מיזוג (merge) של פרומפטים מוצלחים, מותציות ממוקדות על בסיס המשוב, וслиוב אלמנטים מהמועמדים הטובים ביותר.

5. שלב 5: סלקציה (Selection)

- * בחירת המועמדים הטובים ביותר לדור הבא באמצעות עקרון פארטו (Pareto) לאיזון בין מטרות, תוך שמירה על גיוון באוכלוסייה.

חזרה לשלב 2 עד להתכנסות או סיום התקציב.

2.3 8. למה הרפלקציה כל כך חשובה?

אופטימיזציה רגילה	GEPA עם רפלקציה
פרומפט A ← ציון 0.3 פרומפט B ← ציון 0.5 פרומפט C ← ציון 0.4	פרומפט A ← ציון 0.3 משוב: "נכשל בשאלות מרובות-שלבים" פרומפט B ← ציון 0.5 משוב: "הצליח כהוסף חשיבה צעד-אחר-צעדים"
מסקנה: B טוב יותר. אבל למה? לא יודעים. → دور הבא: פרומפט משופר המשלב את התובנות.	רפלקציה: "צריך להוסיף הוראה לחשיבה מובנית ובדיקה עצמית בסוף"

3. חלך ג': פרמטרים של GEPA

3.1. 9. פרמטרי תקציב (Budget Parameters)

GEPA משתמש בגישה "תקציב" במקומם פרמטרים מפורטים כמו מספר סיבובים.

פרמטר	תיואר
auto	הגדרת תקציב אוטומטית. ערכים אפשריים: <ul style="list-style-type: none"> □ "light" - תקציב קטן, ריצה מהירה (מתאים לניסויים וראשוניים). □ "medium" - תקציב בינוני (אייזון בין מהירות ליאות). □ "heavy" - תקציב מקסימלי, התוצאות הטובות ביותר ביותר (מומלץ!).
max_full_evals	חלוקת ל-auto: מספר מקסימלי של הרצות מלאות (rollouts) שהאופטימיזטור יכול לבצע. לשימוש כשרוצים שליטה מדוקית בתקציב.

3.2. 10. פרמטרי מיזוג (Merge Parameters)

פרמטר	תיואר
use_merge	אם להשתמש באופטימיזציה מבוססת מיזוג. ברירת מחדל: True.
max_merge_invocations	מספר מקסימלי של פעולות מיזוג. ברירת מחדל: 5.

3.3. 11. פרמטרי הערכה (Evaluation Parameters)

פרמטר	תיאור
num_threads	מספר threads להערכת מקבילית. ערך גבוה יותר = ריצה מהירה יותר (לדוגמה: 32).
failure_score	הציון שניתן לדוגמאות שנכשלו. ברירת מחדל: 0.0.
perfect_score	הציון המקסימלי האפשרי. ברירת מחדל: 1.0. (לזיהוי מתי הכל מושלם).
track_stats	האם לעקוב ולשמור סטטיסטיות (לניתוח ודיבוב).

3.4 12. פרמטרי מודל הרפלקציה (Reflection LM)

חשוב! מודל הרפלקציה הוא קרייטי להצלחת GEPA! מומלץ לשימוש במודל חזק עם יכולות חשיבה טובות.

* **reflection_lm:** מודל השפה לביצוע הרפלקציה.

4 חלק ד': מושגי יסוד בהגדרת משימה

4.1 13. חתימה (Signature)

החתימה היא הגדרה פורמלית של המשימה. היא מצוינת:

- * **שדות קלט:** מה המודול מקבל (למשל: שאלת).
- * **שדות פלט:** מה המודול צריך להשזר (למשל: תשובה).
- * **תיאור המשימה:** הסבר כללי על מה צריך לעשות.

4.2 14. סט נתונים (Dataset)

סט הנתונים הוא אוסף דוגמאות שמורות למודל מה אנחנו מצפים. **כמות מומלצת:** מינימום 30, מומלץ 50-100, אופטימלי 200+.

טיפ חשוב לחלוקת נתונים ב-DSPy: בנויגוד למידת מכונה קלאסית, ב-DSPy **מומלץ יותר validation מאשר train** עבור אופטימיזורי פרומפטים. הטוווח המומלץ לכל סט הוא 300-30 דוגמאות.

4.3 15. Metric (מדד) - דרישות מיוחדות ל-GEPA

קרייטי! המודד ב-GEPA שונה מממדדים רגילים. הוא חייב:

1. לקבל 5 פרמטרים: pred_trace, pred_name, trace, pred, gold

סס	מטרה
Train	ללמידה מהדוגמאות וליצור פרוומפטים (אימון)
Validation	לבחור בין גרסאות שונות (ומילץ: יותר מ-Train!) (ולידציה)
Test	למדוד ביצועים סופיים על נתונים שלא נראו (מבחן)

2. להציג `dspy.Prediction` עם:

* **score**: ציון מספרי בין 0 ל-1.

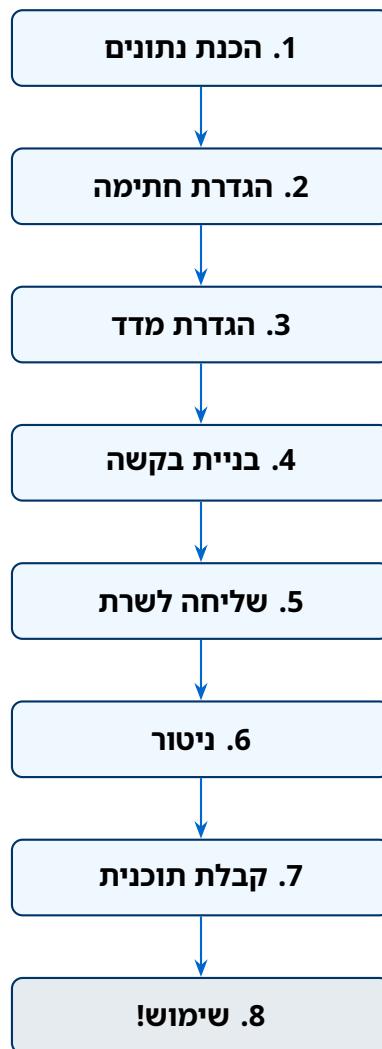
* **feedback**: משוב טקסטואלי מפורט.

דוגמה ללוגיקת מדד:

- * **תשובה זהה לחלווטין**: ציון 1.0, משוב "תשובה מושלמת!".
- * **תשובה חלקית**: ציון 0.5, משוב "תשובה חלקית. הנכוון הוא X, קיבלנו Y. חסר Z".
- * **תשובה שגوية**: ציון 0.0, משוב "שגוי לחלווטין. המודל התבבל בין A ל-B...".

5. חלק ה': זרימת העבודה המלאה

5.1. סקירת התהליך



5.2. מבנה הבקשה לשרת

שדות חובה ואופציונליים בבקשת:

שדה	תיאור
module_name	סוג המודול. לרוב "dspy.ChainOfThought". (מוסיף שדה reasoning לפני הפלט, משפר ביצועים במשימות מורכבות).
optimizer_name	שם האופטימיזר ("A").dspy.GEPA".
signature_code	קוד החתימה (CTKST).
metric_code	קוד המודד (CTKST).

תיאור	שדה
רשימת הדוגמאות (SOSJ).	dataset
מייפוי בין שמות השדות בחתימה לבין שמות העמודות בנתונים (למשל: אם בחתימה יש question ובנתונים q).	column_mapping
הגדירות מודל השפה הראשי.	model_config
הגדירות מודל הרפלקטיבי (חובה ל-GEPA!).	reflection_lm
פרמטרים לאופטימיזר (כמו "heavy" (auto="heavy").	optimizer_kwargs
(אופציונלי) יחס החלוקת ל-Train/Val/Test (למשל [0.2, 0.3, 0.5]).	split_fractions
(אופציונלי) האם לערבב את הנתונים לפני החלוקת.	shuffle
(אופציונלי) זרע אקראיות לשחזור תוצאות (Reproducibility).	seed

5.3 18. רשימת נקודות קצרה (API Endpoints)

להלן הכתובות המרכזיות לעובדה מול השירות:

כתובת (Endpoint)	מטרה
/run	שליחת עבודת אופטימיזציה חדשה.
/jobs/{id}	בדיקה סטטוס כללי של העבודה.
/jobs/{id}/summary	קבלת סיכום תוצאות, ציונים וגרפים.
/jobs/{id}/logs	צפייה בלוגים של הריצה לדיבוג.
/jobs/{id}/artifact	הורדת הקובץ המאופטם הסופי.
/health	בדיקה תקינות השירות.

5.4 19. ניטור התקדמות

*: ממתין בתור.

*: האופטימיזציה רצה CUT.

*: הסתים בהצלחה.

*: נכשל (בדוק לוגים).

6 חלך 1: טיפים ושיטות עבודה מומלצות

6.1 20. טיפים להכנת נתונים

- * איקות מעל חממות (50 מדיקות עדיפות על 200 בינות).
- * גיון (מקרי קצה).
- * ניקיון (לא שגיאות).
- * עקביות ויצוגיות.

6.2 21. טיפים לכתיבה מדדים ל-GEPA

- * **משוב מפורט:** הסבירו בדוחן מה היה לא נכון.
- * **משוב קונסטרוקטיבי:** צינו מה צריך לשפר.
- * **ציון הוגן:** תנו ניקוד חלק.
- * **עקביות:** ציונים דומים למקרים דומים.

6.3 22. פתרון בעיות נפוצות

בעיה	פתרון
העבודה נכשלה מיד	בדקו פורמט בקשה ושגיאות NSJ.
תוצאות גרועות	הוסיפו דוגמאות מגוונות, שפרו את המחשב במדד.
ריצה איטית מאוד	השתמשו ב- <code>-light</code> = <code>auto</code> לניסוי ראשוני, הגדילו <code>threads_num</code> .
המדד תמיד נותן 0	בדקו את לוגיקת השוואה, הוסיפו גמישות (אوتיות קטנות, רווחים).
הרפלקציה לא עוזרת	שדרגו את מודל הרפלקציה, שפרו את המחשב במדד.

7 חלק 2: מילון מונחים

מונח	הסבר
LLM	מודל שפה גדול.
Prompt	הוראה שנשלחת למודול.
DSPy	תוכנה לתוכנות מודלי שפה (Declarative Self-improving Python).
Optimizer	אלגוריתם שմשפר פרומפטים אוטומטיות.
GEPA	ס-Genetic-Pareto - אופטימיזר עם רפלקציה.
Reflection	תהליכי הניתוח של הצלחות/כישלונות.
Signature	הגדרת קלט/פלט של המשימה.
Metric	פונקציה לבדיקת הצלחה ומתן משוב.
Dataset	אוסף דוגמאות לאימון והערכה.
Train / Val / Test	סטים ללמידה, בחירה (ולידציה) ומדידה סופית.
Job	עבודת אופטימיזציה בשרת.
Artifact	התוכנית המכואופטמת הסופית.
Rollout	הרצה מלאה אחת של התוכנית.
Trace	מסלול הביצוע (כל השלבים שהמודול עבר).
ChainOfThought	מודול שמוסיף חשיבה צעד-אחר-צעד.
reflection_lm	מודל השפה שמבצע את הרפלקציה ב-GEPA.